Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Algoritmos de detección de objetos para la detección y seguimiento de ojos

TITULACIÓN: Grado en Ingeniería de Sistemas de Telecomunicaciones

AUTOR: Miguel Ángel Antúnez Galindo

DIRECTOR: Francesc Tarrés Ruiz

FECHA: 29 de enero del 2019

Título: Algoritmos de detección de objetos para el rastreo de ojos

Autor: Miguel Ángel Antúnez Galindo

Director: Francesc Tarrés Ruiz

Fecha: 23 de septiembre del 2018

Resumen

La Dirección General de tráfico ha determinado que la somnolencia es un factor implicado, directa o indirectamente, en entre el 15 y el 30% de los accidentes de tráfico en España [1]. Según el estudio presentado por el Comisariado Europeo del Automóvil (Fundación CEA). "Sueño y fatiga, ¿cuáles son los hábitos de los conductores españoles? [2]" Un 59,22% de los conductores declara haber sufrido micro sueños y un 71,65% de los encuestados ha sentido síntomas de somnolencia al volante. Con los datos proporcionados por la dirección general de tráfico, queda demostrada la necesidad de dotar con un sistema que permita detectar síntomas de somnolencia en los conductores de automóviles.

El proyecto consiste en diseñar una solución de software capaz de identificar mediante el análisis de imágenes indicios de fatiga en el conductor, para ello se implementarán los algoritmos de Viola-Jones y transformada de Hough para la detección y rastreo de ojos. El sistema deberá valorar la fiabilidad de los dos algoritmos seleccionados, y que posteriormente puedan ser implementados en procesadores en tiempo real y utilizarlos en aplicaciones de seguridad en conducción. Los objetivos son establecer el coste computacional y determinar la fiabilidad de los algoritmos y en futuros estudios poder implementar el algoritmo en un dispositivo móvil.

Para llevar a cabo el análisis y la implementación de los algoritmos, se utilizará la herramienta matemática Matlab, una vez desarrollada la implementación, se podrán a prueba la fiabilidad del sistema con un dataset público de diferentes imágenes de personas para valorar distintos escenarios fuera del entorno de pruebas.

Title: Object detection algorithms for the detection and monitoring of eyes

Author: Miguel Ángel Antúnez Galindo

Director: Francesc Tarrés Ruiz

Date: January 29, 2019

Overview

The Directorate General of Traffic has determined that sleepiness is a factor involved, directly or indirectly, in between 15 and 30% of traffic accidents in Spain [1]. According to the study presented by the European Commission of the Automobile (CEA Foundation). "Sleep and fatigue, what are the habits of Spanish drivers?" [2]. A 59.22% of drivers say they have suffered micro dreams and 71.65% of respondents have felt symptoms of drowsiness behind the wheel. With the data provided by the general direction of traffic, it is demonstrated the need to provide a system to detect symptoms of sleepiness in car drivers.

The project consists in designing a software solution capable of identifying by means of the analysis of images symptoms of fatigue in the driver, for this purpose the algorithms of Viola-Jones and Hough transform will be implemented for the detection and tracking of eyes. The system should evaluate the reliability of the two algorithms selected and that can later be implemented in processors in real time and used in driving safety applications. The objectives are to establish the computational cost and determine the reliability of the algorithms and in future studies to be able to implement the algorithm in a mobile device.

To carry out the analysis and implementation of the algorithms, the Matlab mathematical tool will be used, once the implementation is developed, the reliability of the system can be tested with a public dataset of different images of people to assess different scenarios outside the environment of tests.

INDICE

INTRODUCCIÓN

El sueño al volante es una circunstancia que se relaciona con una buena cantidad de accidentes de tráfico. Los accidentes debidos a quedarse dormido al volante suelen ser mortales ya que se inhiben por completo los sentidos del conductor, que se ve desprovisto de todo control sobre el vehículo. Según el estudio presentado por el Comisariado Europeo del Automóvil (Fundación CEA) [2]. Un 59,22% de los conductores declara haber sufrido micro sueños y un 71,65% de los encuestados ha sentido somnolencia al volante. El estudio además se ha centrado en analizar los hábitos de los conductores durante los viajes, el 49,93% sólo descansa cuando se siente cansado y todavía un 4,58% no para en todo el viaje, con lo que se incrementan las posibilidades de accidente.

Algunas marcas de coche ponen a disposición el detector de fatiga, dicho detector consiste en unos sensores que interpretan los movimientos del volante y recomiendan al conductor que se tome un pequeño descanso si detecta anomalías en su conducción, por norma general las compañías no ponen a disposición este sistema en sus versiones más económicas, es evidente que con la oferta que hay actualmente a disposición de los conductores estándar, hay una necesidad que cubrir en términos de seguridad en los vehículos que actualmente se comercializan.

Para intentar solucionar el evidente problema, nace la idea de aprovecharse de una de las herramientas a disposición de casi cualquier persona, los Smartphone, se prevé que para el año 2020 el 70% de la población disponga de un teléfono inteligente, por este motivo sería conveniente que los conductores dispusieran de un sistema de alerta accesible y democratizado ante un posible accidente por somnolencia.

La motivación para la realización de este proyecto viene dada en gran parte por las crecientes posibilidades del procesado y análisis de imágenes, que se pueden llevar a cabo mediante hardware de bajo coste en multitud de entornos. La mayoría de las técnicas de procesado de imágenes implican tratar la imagen como una señal bidimensional y aplicarle técnicas de procesamiento de señal estándar, por lo que una de las herramientas de software matemático más potentes como lo es Matlab y específicamente su librería Image Processing Toolbox, se convierte en una herramienta de gran versatilidad para el desarrollo de este proyecto ya que nos proporciona un conjunto completo de algoritmos estándar de referencia para el análisis y la visualización de imágenes. Matlab es un buen lenguaje para el desarrollo de los algoritmos, proporciona herramientas para valorar el coste computacional y la fiabilidad del sistema, pero en la práctica no es eficiente en temas de rendimiento, por lo que en estudios futuros de debe buscar lenguajes más eficientes a la hora de implementar la aplicación en un entorno real.

En este estudio se busca allanar el camino hacia una solución de software, que, mediante el procesado de imágenes, permita monitorizar rostro y ojos del conductor, para detectar el cierre de sus ojos seleccionando solo esa parte de la cara, obteniendo imágenes en vivo usando una cámara de bajo coste, para, de la forma más inmediata posible, alertarle con el fin de evitar un posible accidente de tráfico. Si se consigue evaluar o implementar un algoritmo que sea lo suficientemente fiable y presente un buen coste computacional, se podría implementar una aplicación “low cost” que solucione el problema presentado. La primera implementación consiste en evaluar una serie de algoritmos de detección de ojos. El objetivo es determinar el algoritmo que implique menor coste computacional y mayor eficiencia en este reconocimiento y que pueda ser implementado en terminales de bajo costo, en pocas palabras, proporcionar de una herramienta de seguridad automovilística a cualquier usuario estándar.

El proyecto está dividido en 3 capítulos principales que se detallan a continuación.

Algoritmos de detección de objetos

El capítulo consiste en una explicación teórica extendida de los dos algoritmos implementados, define el funcionamiento paso a paso del conjunto de elementos que forman dichos algoritmos, asienta las bases para la correcta comprensión de la implementación de software, y expone las razones fundamentales que convierten a estos dos detectores de objetos en soluciones válidas para la implementación de una aplicación fiable, en la detección y rastreo de ojos.

Implementación en Matlab

Una vez expuestos los conceptos básicos de los algoritmos de detección de objetos, este capítulo explica el desarrollo e implementación de los algoritmos en Matlab, a través de una solución de software se pone en evidencia los aspectos positivos y negativos de cada algoritmo, con ejemplos prácticos dentro de un entorno de pruebas controlado para cada implementación.

Para garantizar que los resultados obtenidos son extrapolables a un entorno real, en dicha sección se pone a prueba la implementación conjunta con un dataset público de diferentes individuos.

Discusión de los resultados

Discusión del comportamiento de los algoritmos en las mismas condiciones dentro de un entorno de pruebas. Se discute las razones fundamentales por las cuales un algoritmo puede ser más robusto a cambios en el entorno, y la razón por las cuales uno presenta mejor rendimiento en el tiempo de procesado de imagen.

Se aporta literatura de las herramientas de desarrollo para la implementación a futuro de la aplicación en un terminal móvil.

CAPÍTULO 1. ALGORITMOS DE DETECCIÓN DE OBJETOS

La detección de objetos es una tecnología relacionada con la visión artificial y el procesamiento de imágenes que trata de detectar casos de objetos semánticos de una cierta clase (como humanos, edificios, o coches) en vídeos e imágenes digitales.

Cada clase de objeto tiene sus propias características especiales que ayuda en la clasificación de clase, por ejemplo, todos los círculos son redondos, la detección de clase del objeto utiliza estas características especiales, cuándo se buscan círculos, se examinan los objetos que están a una distancia particular de un punto (el centro), de un modo parecido, cuándo se buscan cuadrados, se necesitan objetos que son perpendiculares en las esquinas y tienen lados con la misma longitud. Una aproximación similar se emplea para identificación facial donde ojos, nariz, y los labios pueden ser identificados, al igual que características como el color de piel y la distancia entre los ojos.

1.1 Viola-Jones

El algoritmo de Viola-Jones describe un sistema para la detección de objetos en imágenes a tiempo real. Este algoritmo tiene un coste computacional muy bajo, junto con una probabilidad de verdaderos positivos del 99,9% y de falso positivos del 3,33% [3] característica especialmente significativa para cumplir unos de los requisitos fundamentales de este proyecto, que es el de dotar de una gran fiabilidad a la hora de detectar la cara de un conductor.

La motivación de Viola y Jones para la creación de este algoritmo fue la búsqueda de una rápida detección de caras, pero puede ser usado para la detectar cualquier objeto, por lo que resulta un recurso muy valioso en la detección del rostro y los ojos del conductor de un automóvil. Según sus creadores, este algoritmo es capaz de detectar caras en una imagen de 384x288 píxeles a una velocidad de 15 imágenes por segundo utilizando un procesador Pentium III a 700 MHz [5], en la implementación que se propone en este proyecto se ha obtenido de media 7 imágenes por segundo, utilizando imágenes de 640x360 pixeles en un procesador Intel Core i7-6700HQ de 2.60 GHz.

La detección de rostros es una tarea que requiere mucho tiempo, especialmente a medida que aumenta el tamaño de la imagen que se procesará. Hasta la fecha, se ha reportado mucho trabajo en la literatura en un intento por acelerar el proceso de detección de objetos, particularmente en la cara. Esto es para satisfacer la demanda actual de aplicaciones como el sistema de video vigilancia en tiempo real. Por ejemplo, Theocharides et al. [6] propuso una arquitectura ASIC que explota fuertemente el paralelismo del algoritmo Viola-Jones al paralelizar los accesos de datos de imagen. Como resultado, muestra una tasa de cálculo de 52 fps, aunque en él estudio no se mencionan las resoluciones de la imagen de entrada.

En los últimos años, la computación GPU de NVidia se está volviendo popular para el uso de la paralelización debido a su arquitectura. Ya hay varios trabajos relacionados con la aceleración del algoritmo de detección de rostros Viola-Jones utilizando CUDA (plataforma de computación en paralelo). Por ejemplo, las implementaciones principales de GPU en el algoritmo se ejecutan a 2.8 fps en un NVidia GTX285, y a 4.3 fps en dos NVidia GTX295 para imágenes de resolución 640x480 [7]. Li-chao Sun et. al [8] presentó un clasificador en cascada Viola-Jones basado en un sistema de detección de rostros en tiempo real en la plataforma CUDA. Los resultados experimentales muestran que el programa CUDA que se ejecuta en una tarjeta gráfica Nvidia GTX 570 para una imagen de entrada VGA podría alcanzar velocidades 6 veces mayor con 9.35 fps en comparación con la versión de una sola CPU. Además, Shivashankar J. Bhutekar et. al [15] propuso una técnica que procesa la imagen para la detección y reconocimiento de la cara en paralelo en la GPU NVIDIA GeForce GTX 770. El algoritmo de detección de rostros Viola-Jones muestra aproximadamente 3 fps en una imagen de 700x580 píxeles en el marco CUDA, en comparación con 0.71 fps en la implementación de una sola CPU.

Los estudios antes mencionados presentaron sus resultados en términos de velocidad. Sin embargo, otra configuración paramétrica, como el factor de escala, el tamaño mínimo y el control por pasos de la ventana de búsqueda de su detector, influye en el rendimiento de detección en términos de velocidad y precisión. Por lo tanto, se necesita una cuidadosa consideración tanto de las condiciones de configuración paramétrica como de la velocidad de cuadros para justificar con precisión el rendimiento de cualquier sistema de detección de rostros.

A diferencia de otros algoritmos utilizados en la detección de objetos, Viola-Jones procesa sólo la información presente en una imagen en escala de grises. No utiliza directamente la imagen, sino una representación llamada imagen integral [3]. Para determinar si en una imagen se encuentra el objeto buscado o no, el algoritmo divide la imagen integral en subregiones de tamaños diferentes y utiliza una serie de clasificadores (clasificadores en cascada), cada una con un conjunto de características visuales. La utilización de este algoritmo supone un ahorro de tiempo considerable ya que no serán procesadas subregiones de la imagen que se sepa con certeza que no contienen el objeto buscado y sólo se invertirá recursos en aquellas subregiones que posiblemente si lo contengan. A continuación, se procede a analizar paso a paso el funcionamiento del algoritmo para determinar su utilidad para el objetivo de este proyecto.

1.1.1 Filtros tipo Haar

Los filtros tipo Haar son similares a los filtros de detección de contornos, reciben este nombre por similitud a los filtros utilizadas en los wavelet de Haar y se utilizan para detectar la presencia de ciertas características en una imagen.

Se utilizan tres tipos de filtros que dan un único valor como resultado. El valor de un rectángulo se define como la diferencia entre la suma de pixeles en la región blanca y la suma de pixeles en la región gris.

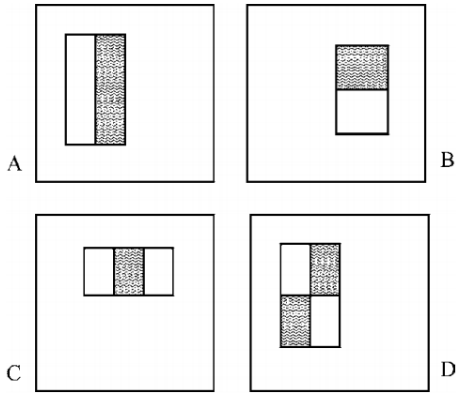


Ilustración 1: En (A) y (B) se pueden ver filtros de dos rectángulos. en (C) de 3 rectángulos y en (D) de 4 rectángulos

Los filtros se definen sobre una ventana de búsqueda básica de 24x24 píxeles, el tamaño y la posición de una ventana pueden variar siempre que sus rectángulos en blanco y negro tengan la misma dimensión, se unan entre sí y mantengan sus posiciones relativas. Gracias a esta restricción, el número de características que se pueden dibujar de una imagen es algo manejable [4].

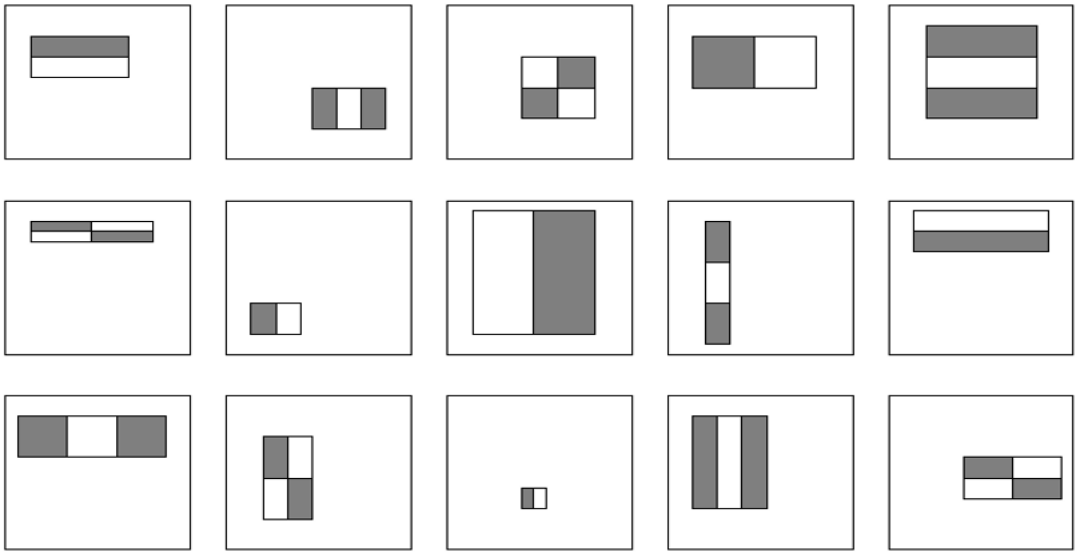


Ilustración 2: Diferentes posibilidades de filtros

Considerando todos los posibles parámetros de los filtros, posición, escala y tipo, da lugar a más de 160,000 resultados posibles, estos resultados se obtienen utilizando la imagen integral.

1.1.2 Imagen integral

Para reducir el número de operaciones, puesto que hay que aplicar estos filtros numerosas ocasiones por imagen, en lugar de sumar todos los pixeles bajo cada uno de los bloques del filtro, Viola y Jones idearon un sistema llamado imagen integral, cuyo resultado es la suma de todos los pixeles dentro del recinto que se extiende por arriba y por la izquierda de cada uno de los pixeles de la imagen.

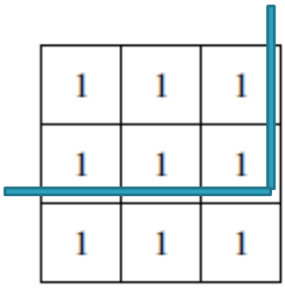


Ilustración 3: imagen original

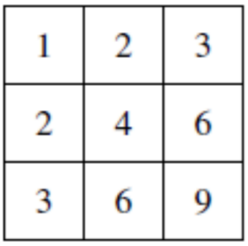


Ilustración 4: imagen integral

Esta técnica permite calcular la suma de todos los pixeles dentro de un rectángulo usando solo los valores de las 4 esquinas del rectángulo.

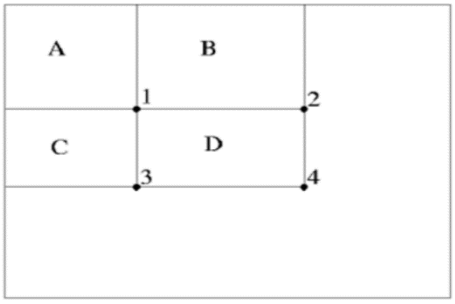


Ilustración 5: representación de una imagen integral

La suma de los pixeles dentro del rectangulo D se puede calcular con solo 4 valores de referencia. El valor del punto 1 de la imagen integral de la figura es la suma de todos los pixeles del rectangulo A. el valor del punto dos es la suma de los rectangulos A y B, la posición 3 es la suma de A y C, finalmente el valor de la posición 4 equivale a la suma de A, más B, más C, más D. Por lo tanto la suma de todos los pixeles bajo el rectangulo D se puede calcular como el valor de la posiciones 4 + 1 – (2 + 3) de la imagen integral.

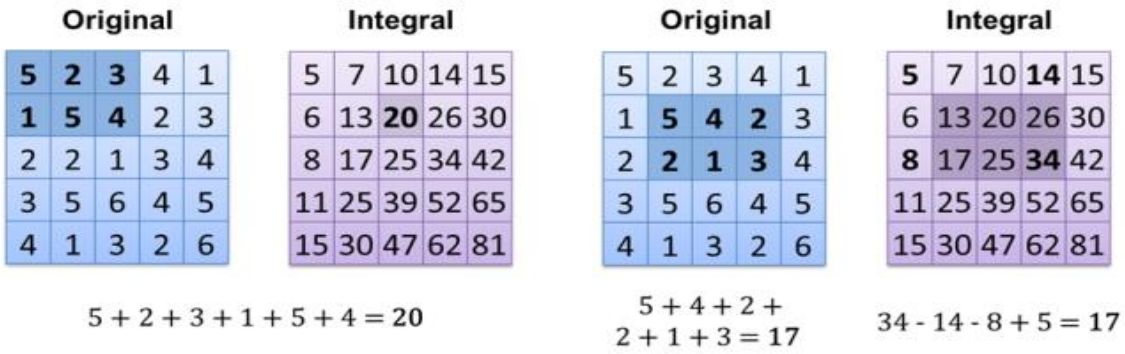


Ilustración 6: Ejemplos del cálculo de la suma de pixeles bajo una región

1.1.3 Aprendizaje con Adaboost

Antes de crear los clasificadores en cascada es necesario realizar un proceso de entrenamiento supervisado. Este entrenamiento se realiza mediante un algoritmo basado en AdaBoost, un algoritmo adaptativo de machine learning cuyo nombre es una abreviatura de adaptive boosting.

Dado un conjunto de filtros y un conjunto de imágenes de entrenamiento (positivas y negativas), se usa una variante de AdaBoost para seleccionar un pequeño conjunto de filtros y entrenar al clasificador. En su forma original, el algoritmo de aprendizaje de AdaBoost se utiliza para potenciar el rendimiento de clasificación de un algoritmo de aprendizaje simple.

Al tener más de 160,000 funciones de filtro rectangular asociadas a cada sub-ventana de imagen, es decir, una cantidad mucho más grande que el número de píxeles por cada bloque de 24x24, aunque cada filtro se puede calcular de manera muy eficiente mediante la imagen integral, calcular el conjunto completo conlleva un coste computacional prohibitivo, por lo que la hipótesis de Viola y Jones (confirmada mediante la demostración experimental) es que existe un número muy pequeño de estos filtros que, combinados, forman un clasificador eficaz. El principal desafío es encontrar estos filtros. Por ello el algoritmo de aprendizaje está diseñado para seleccionar los filtros que discriminen mejor las imágenes, dado un conjunto de imágenes previamente etiquetadas con ejemplos positivos y ejemplos negativos.

Ilustración 7: filtro relevante

Ilustración 8: filtro irrelevante

Después de que se encuentran estos filtros relevantes, se usa una combinación ponderada de todos estos filtros para evaluar y decidir si una sub-ventana determinada contiene tiene el objeto buscado o no. Un filtro se incluye en el clasificador si en más de la mitad de los casos detecta el objeto buscado, estos filtros cuyo resultado es ponderado, también se llaman clasificadores débiles.

Para seleccionar filtros se entrenan clasificadores débiles limitados a usar un único filtro, para cada uno de ellos, el clasificador débil determina el valor umbral que minimiza los ejemplos mal clasificados. Un clasificador débil $h_j(x)$ por tanto consiste en un filtro f_j, un valor umbral θ_j y un coeficiente p_j indicando la dirección del signo de desigualdad.

$$h_j(x) = \begin{cases} 1 & si\ p_j f_j(x) < p_j \theta_j \\ 0 & e.o.c \end{cases}$$

Inicialmente en AdaBoost se parte de un conjunto de imágenes $(x_1, y_1), \ldots, (x_n, y_n)$ donde y_i es igual a 0 para ejemplos negativos e igual a 1 para ejemplos positivos.

Se inicializan los pesos $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_i = 0$, e $y_i = 1$ respectivamente, donde m es el número de negativos y l el número de positivos.
Para cada iteración, $t = 1, \ldots, T$:

- Se normalizan los pesos de las ponderaciones

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

- Para cada filtro, j, se entrena un clasificador h_j que solo utiliza un filtro, el error se evalúa teniendo en cuanta los pesos wt , ϵj $= \sum$ wi |hj (xi) − yi |
- Se elije el clasificador h_t , con menor error ϵ_t
- Se actualizan los pesos:

  $w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$

- Donde $e_i = 0$ si el ejemplo de x_i se clasifica correctamente y $e_i = 1$ en caso contrario y βt $= \frac{\epsilon_t}{1-\epsilon_t}$
- El clasificador fuerte final es:

$$h(x) = \begin{cases} 1 & si \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & e.o.c \end{cases}$$

  donde $\alpha_t = \frac{1}{\beta_t}$.

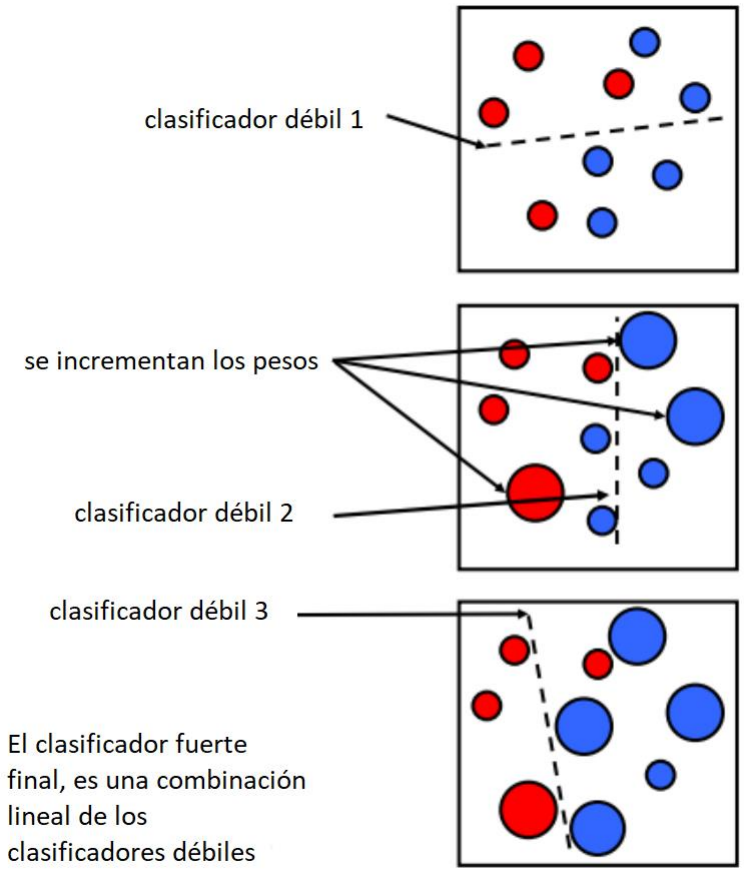


Ilustración 9: Representación orientativa del cálculo de un clasificador fuerte

1.1.4 Clasificación en cascada

El principio básico del algoritmo de Viola-Jones es el escaneo de la misma imagen múltiples veces con el mismo clasificador, pero en distintos tamaños de la imagen, para poder encontrar el objeto buscando en cualquiera de las dimensiones en las que este pueda aparecer en la imagen. Mediante este proceso, incluso si una imagen contiene una o más veces el objeto buscado, fotografiado desde distintas distancias, el algoritmo es capaz de detectarlo.

El inconveniente de sobre escanear una imagen (aparte del coste computacional) es la gran cantidad de falsos positivos que genera. Así, el algoritmo debe concentrarse en descartar rápidamente las regiones que no contienen el objeto buscado para invertir más a recursos en las subregiones con posibles positivos. Se usa un clasificador en cascada compuesto por etapas, donde cada etapa contiene un clasificador fuerte y donde los filtros se encuentran agrupados en las distintas etapas, donde cada una de estas determina si una sub-ventana contiene o no contiene el objeto buscado. De manera que una sub-ventana identificada como no contenedora del objeto buscado, es descarta inmediatamente, en cambio, si puede contener el objeto buscado continua hacia la siguiente etapa. De esta forma se obtiene una cascada de clasificadores, donde cada uno es entrenado con AdaBoost y sus valores umbrales se ajustan para minimizar los falsos negativos. La cascada de clasificadores implementada

por Viola y Jones para la detección de rostros consta de 38 etapas y más de 6000 filtros, pero de media se evalúan únicamente 10 filtros por ventana de búsqueda.

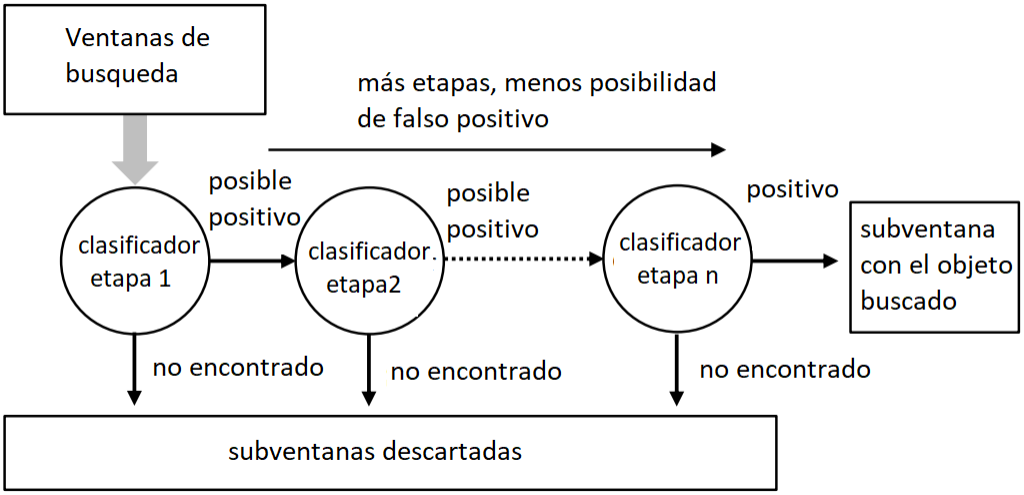


Ilustración 10: Esquema del clasificador en cascada del algoritmo de Viola-Jones

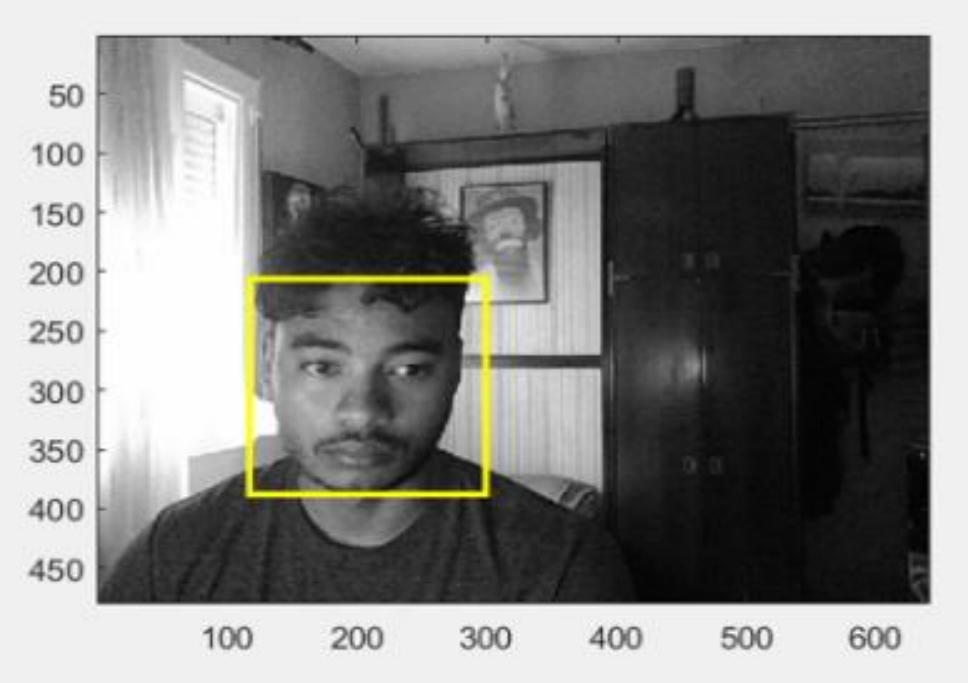


Ilustración 11: Sub-ventana con el objeto buscado (rostro) detectado

El compuesto de todos estos elementos construye un algoritmo muy fiable a la hora de detectar un objeto, siempre y cuando este no tome variaciones muy bruscas. El algoritmo Viola-Jones tiene problemas a la hora de detectar variaciones en la escala o rotación del objeto deseado. En el capítulo 2 se implementará una solución de software del conjunto de todos los componentes anteriormente descritos en este capítulo.

1.2 Transformada de Hough circular

La transforma de Hough es un algoritmo que permite detectar formas geométricas que puedan describirse con ecuaciones paramétricas sencillas, como rectas, parábolas, círculos, etc.

1.2.1 Detección de contornos

Primeramente, se aplica un algoritmo de detección de contornos, como el algoritmo de Canny, para encontrar la silueta de cualquier objeto que posea algún contraste con su fondo o con otra superficie adyacente, este proceso es de esencial para conseguir una imagen binaria, es decir, con solo negros y blancos, que facilita enormemente la identificación por similitud de estas formas con las funciones de formas geométricas simples.

Ilustración 12: Fotograma de un ojo tras aplicar algoritmo de Canny de detección de contornos

Pero este contorno no consta de una única línea de pixeles, sino que se trata de una región entera, gruesa e irregular, con una especie de ruido añadido a la forma descrita por la función paramétrica, que en el caso del circulo es:

$$(x-a)^2 + (y-b)^2 = R^2$$

1.2.2 Búsqueda de círculos

La transformada Hough se puede usar para determinar los parámetros de un círculo cuando se conocen varios puntos que están en el perímetro. Puede describir un círculo con radio (R) y centro (a, b) con las ecuaciones paramétricas

$$x = a + R\cos(\theta)$$
$$y = b + R\sin(\theta)$$

Cuando el ángulo θ recorre los 360 grados, los puntos (x, y) trazan el perímetro de un círculo. Si una imagen contiene muchos puntos, algunos de los cuales caen en el perímetro algún círculo, la tarea del algoritmo es encontrar los parámetros (a, b, R) para describir cada círculo. El hecho de que el espacio de

parámetros sea de tres dimensiones hace que la implementación directa de la técnica de Hough pueda requerir un alto coste computacional.

1.2.2.1 Búsqueda de círculos de radio conocido

Si se conoce el radio (R) del círculo la cuestión se reduce a un problema de dos incógnitas. El objetivo es encontrar las coordenadas (a, b) del centro de un círculo.

$$x = a + R\cos(\theta)$$
$$y = b + R\sin(\theta)$$

La ubicación de los puntos (x, y) es el espacio de puntos del perímetro del círculo buscado, sobre el que se van dibujando nuevos círculos (de radio R y centro x, y) a medida que se incrementa θ, por lo que el punto central del círculo buscado (a, b) será la intersección de todos los círculos con centro en el perímetro del círculo buscado.

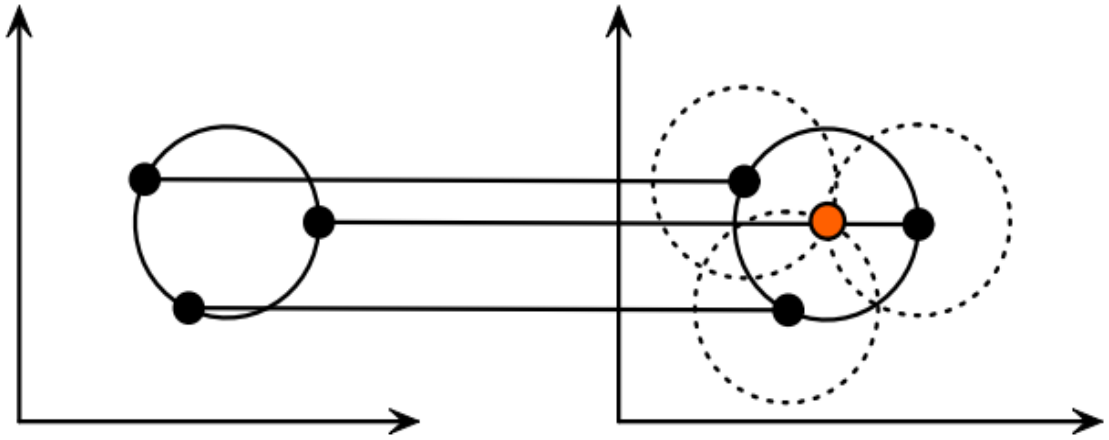


Ilustración 13: Cada punto del perímetro (izquierda) genera un círculo, estos círculos se cruzan en (a, b) que es el centro del circulo buscado

1.2.2.2 Búsqueda de círculos de radio desconocido

Si no se conoce el radio, los puntos del perímetro del circulo buscado generaran círculos descritos por el corte seccional de un cono. Cada punto (x, y) en el perímetro de un círculo producirá una superficie cónica. La a, b y R correctas serán aquellas donde se crecen las superficies de los conos, que construirán un círculo con un radio diferente en cada nivel, r. En este caso el problema posee tres incógnitas y aumentará el coste computacional de la detección.

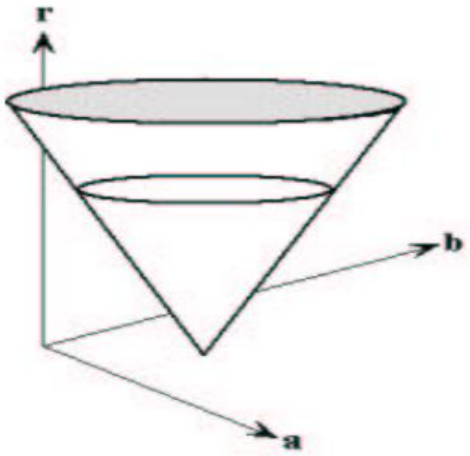


Ilustración 14: Superficie cónica en el espacio, para un punto (x, y)

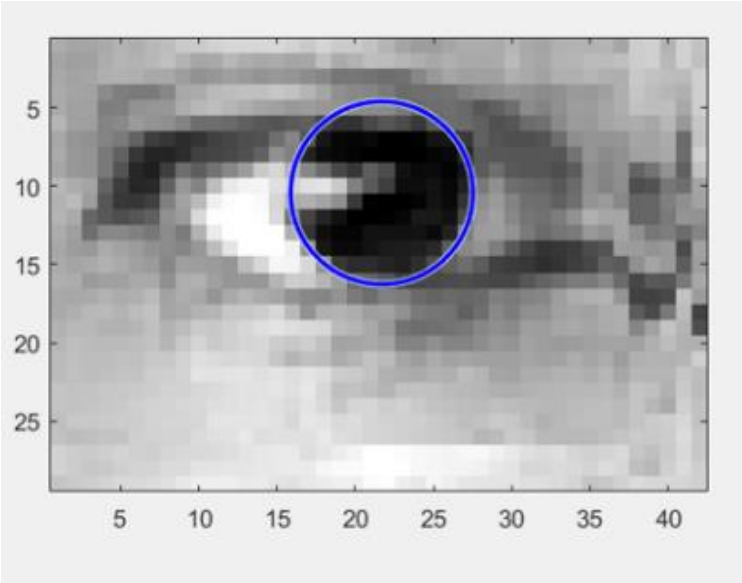


Ilustración 15: Fotograma en escala de grises tras la detección de círculos de Hough con el iris correctamente detectado como círculo

Como el dispositivo de recolección de imágenes estará en una posición relativamente fija respecto la cara del conductor, se puede acotar un rango de posibles radios para minimizar el coste computacional y evitar la detección de círculos no correspondientes al iris. También se puede configurar un umbral a partir del cual, si el círculo está parcialmente cubierto, este ya no se detecta.

La transforma de Hough es una buena alternativa para detectar objetos circulares, y puesto que el iris es una membrana circular y posee un contraste grande con el blanco del ojo, la detección de contornos se puede llevar a cabo exitosamente y él proceso puede aplicarse para detectar la presencia (o no presencia) del círculo del iris, incluso si este está parcialmente cubierto por el parpado. Para corroborar esto, en el capítulo 2 se implementará una solución de software del algoritmo descrito en este capítulo.

CAPÍTULO 2. IMPLEMENTACIÓN EN MATLAB

Uno de los objetivos que se han establecido en capítulos anteriores es el de desarrollar una solución de software capaz de implementar los algoritmos anteriormente descritos, para ello se necesita de un lenguaje capaz de identificar y extraer información significativa de imágenes. MATLAB es un sistema algebraico computacional que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio (lenguaje M). Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

Matlab pone a disposición un conjunto de herramientas para el análisis y procesado de imágenes. La solución que se propone en este proyecto se basa en utilizar la herramienta Computer Vision System Toolbox la cual se puede descargar desde la propia aplicación de Matlab. Este capítulo da las pautas para la implementación del código de los algoritmos Viola-Jones y transformada de Hough, expone los aspectos positivos y negativos según la experiencia propia fuera del marco teórico de los algoritmos, y se detalla el rendimiento tanto individual como conjuntamente de los dos procesos, una vez obtenida una aplicación que funcione en el entorno de pruebas establecido, se debe proporcionar información objetiva e independiente sobre la calidad de la aplicación, para ello se ejecutará el algoritmo con un conjunto de imágenes de entrada, y se evaluara su comportamiento para determinar si la solución obtenida es fiable.

2.1 Viola-Jones

El software Computer Vision System Toolbox incorpora una serie de modelos de clasificación utilizando el algoritmo Viola-Jones para la detección de un objeto en específico. La incorporación de dichos modelos dentro de la herramienta se ejecuta en una serie de pasos que se definen a continuación.

2.1.1 Modelo de clasificación de entrenamiento

Los modelos de clasificación se entrenan mediante la extracción de características de un conjunto de imágenes conocidas. Estas características extraídas se incorporan a un algoritmo de aprendizaje para entrenar el modelo de clasificación.

El tamaño de imagen utilizado para entrenar a los clasificadores define la región más pequeña que contiene el objeto. Los tamaños de las imágenes de entrenamiento varían según la aplicación, el tipo de objeto y las imágenes positivas disponibles. algunos de los modelos utilizados en este proyecto se describen a continuación.

Modelo de clasificación	Tamaño imagen usado para entrenar el modelo	Descripción del modelo
'FrontalFaceCART'	[20 20]	Detecta caras en posición vertical y orientadas hacia adelante. Este modelo está compuesto por clasificadores débiles, basados en la clasificación y el análisis del árbol de regresión (CART). Estos clasificadores utilizan las funciones de Haar para codificar las características faciales.
'FrontalFaceLBP'	[24 24]	Detecta caras en posición vertical y orientadas hacia adelante. Este modelo está compuesto por clasificadores débiles. Estos clasificadores utilizan patrones binarios locales (LBP) para codificar rasgos faciales. Las características de LBP pueden proporcionar robustez contra la variación en la iluminación.
'LeftEyeCART' 'RightEyeCART'	[20 20]	Detecta el ojo izquierdo y derecho por separado. Los clasificadores débiles que componen estos modelos son arboles de regresión y clasificación (CART). En comparación con los grupos de decisión, los clasificadores basados en árboles de CART son más capaces de modelar dependencias de orden superior.
'LeftEye' 'RightEye'	[12 18]	Detecta el ojo izquierdo y derecho por separado. Estos modelos están compuestos por clasificadores débiles. Estos clasificadores utilizan las funciones de Haar para codificar los detalles.

2.1.2 Clasificador en cascada

Este objeto utiliza una cascada de clasificadores para procesar de manera eficiente las regiones de imagen para la presencia de un objeto de destino. Cada etapa de la cascada aplica clasificadores binarios cada vez más complejos, lo que permite al algoritmo rechazar rápidamente regiones que no contienen el objetivo. Si el objeto deseado no se encuentra en ninguna etapa de la cascada, el detector rechaza inmediatamente la región y finaliza el procesamiento. Al terminar, el objeto evita invocar clasificadores que requieren un uso intensivo de cómputo más adelante en la cascada.

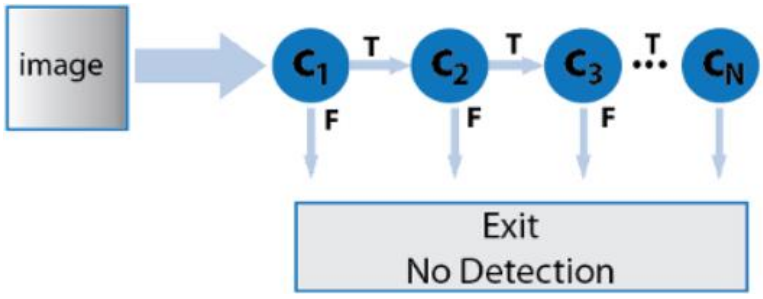


Ilustración 16: Clasificador en cascada utilizado en la herramienta Computer Vision System Toolbox

2.1.3 Detección de objetos multiescala

El detector escala de forma incremental la imagen de entrada para ubicar objetos de destino. En cada incremento de escala, una ventana deslizante, cuyo tamaño es el mismo que el tamaño de la imagen de entrenamiento, escanea la imagen escalada para localizar objetos. La propiedad ScaleFactor determina la cantidad de escala entre incrementos sucesivos por defecto su valor es 1.1.

El tamaño de la región de búsqueda está relacionado con ScaleFactor de la siguiente manera:

Región de búsqueda = round ((ObjectTrainingSize) * (ScaleFactor*N))
N es el incremento actual, un entero mayor que cero, y ObjectTrainingSize es el tamaño de imagen utilizado para entrenar el modelo de clasificación.

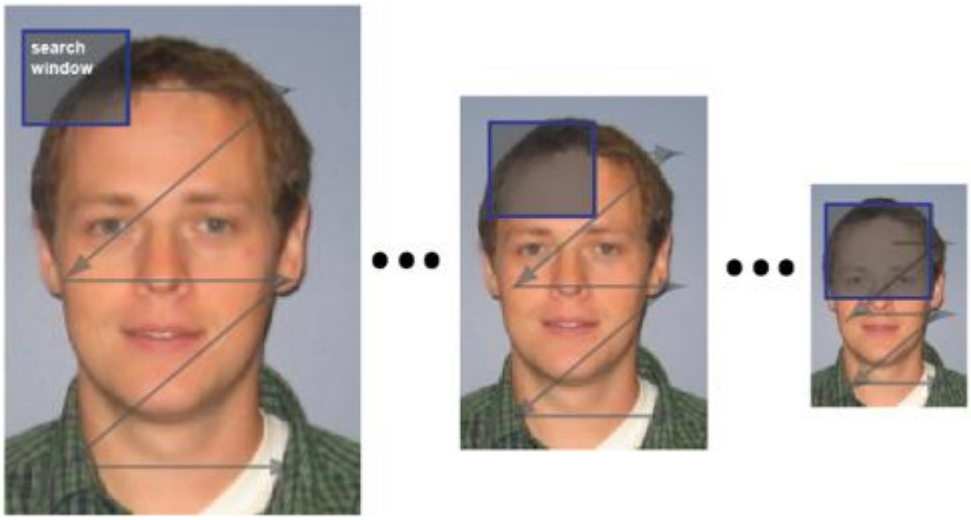


Ilustración 17: Detección de objetos Viola-Jones Matlab

2.1.4 Umbral de detección

Para cada incremento de escala, la ventana de búsqueda atraviesa la imagen produciendo múltiples detecciones alrededor del objeto requerido. Las múltiples detecciones se combinan en un cuadro delimitador por objeto de destino. El algoritmo que facilita Matlab permite utilizar la propiedad MergeThreshold para controlar el número de detecciones requeridas antes de combinar o rechazar las imágenes.

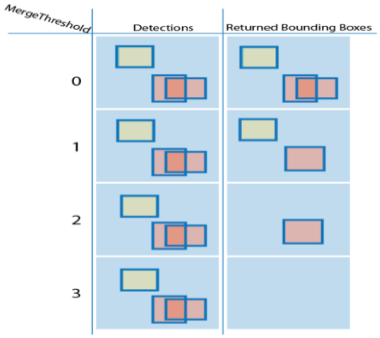


Ilustración 18: Objetos retornados según el umbral

2.1.5 Implementación

Tras utilizar el algoritmo de Viola-Jones en Matlab (cuyo código de implementación se puede consultar en el anexo 6.1) para detectar ojos de una imagen procedente de una webcam (cuyas características se pueden consultar en el anexo 6.5) se ha confirmado la alta fiabilidad en la detección de la región ocular, con un índice bajísimo de falsos positivos, el objeto buscado se detecta perfectamente utilizando casi cualquier tamaño de imagen de entrada.

La implementación desarrollada se basa en el reconocimiento de subregiones faciales. Se declaran los modelos de clasificación 'LeftEyeCART' y 'RightEyeCART', posteriormente se entra en un bucle infinito en donde se procesan las imágenes. Primero se captura con la cámara, la imagen capturada se somete a un conjunto de procesos para adecuarla a la entrada que espera el algoritmo de detección, se procede a hacer la llamada al algoritmo pasándole la imagen previamente procesada, el algoritmo retorna un objeto de detección que contiene las coordenadas exactas en donde se encuentra nuestro objetivo, estos pasos se pueden observar de manera gráfica en el siguiente diagrama de flujo.

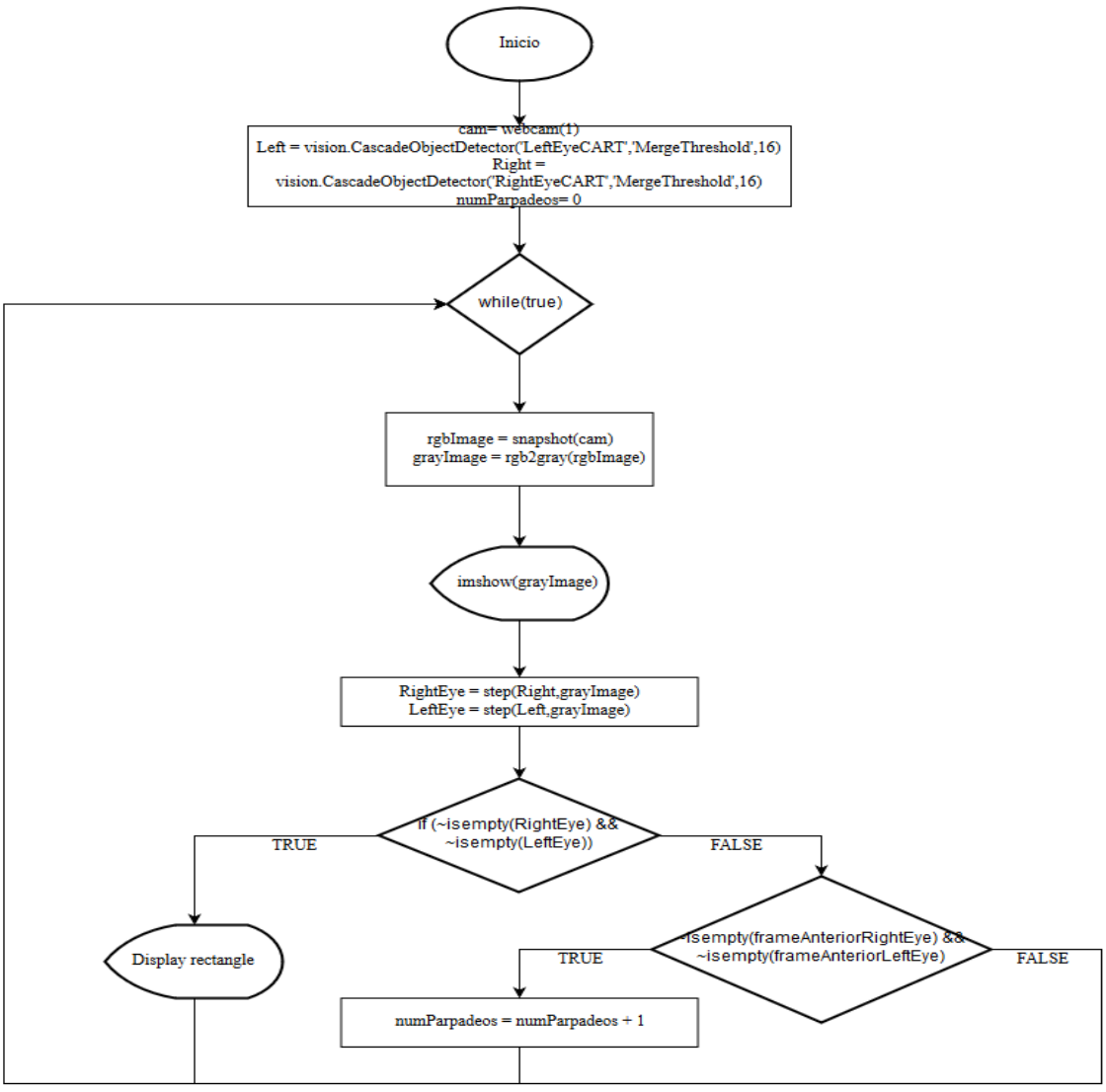
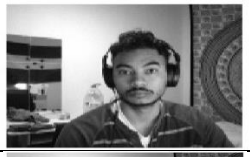


Ilustración 19: Diagrama de flujo algoritmo Viola-Jones

La siguiente tabla muestra el comportamiento del algoritmo según la calidad de la imagen de entrada y el umbral óptimo para cada resolución, a medida que la imagen empeora el umbral debe disminuir, esto también es extrapolable a las condiciones de iluminación del entorno, cuanto más ruido haya en la imagen de entrada menor ha de ser el umbral de detección.

Resolución	Umbral de detección	Imagen
160x120	~	
320x180	4	
320x240	9	
424x240	10	
640x360	20	
640x480	30	

En la implementación del algoritmo se ha optado por los modelos 'LeftEyeCART' y 'RightEyeCART' ya que son mucho mas robustos a rotaciones y movimientos que pueda tener el conductor [9], el tamaño de la imagen que se ha empleado es de 640x360 pixeles con un umbral de detección de 15. Los resultados obtenidos son prometedores, el algoritmo es capaz de detectar el parpadeo y la perdida de la vista sobre el foco principal.

Ilustración 20: Escenarios en donde el algoritmo no detecta los ojos

Utilizando la herramienta de medida de tiempo de ejecución que proporciona Matlab, se puede obtener el tiempo que toma cada línea de código y el número de veces que se llama a la sentencia dentro del código, asi pues, se puede calcular de forma sencilla la tasa de imágenes que es capaz de procesar el algoritmo.

Número de línea	***Función***	***Llamadas***	***Tiempo total (segundos)***	***% de tiempo***
13	RightEye = Right.step(grayImag...	424	20.598	34.2%
14	LeftEye = Left.step(grayImage)...	424	19.555	32.5%
11	imshow(grayImage);	424	11.772	19.6%
9	rgbImage = snapshot(cam);	425	5.981	9.9%
3	cam.Resolution = '640x360';	1	0.835	1.4%
Resto de líneas			1.460	2.4%
Totales			**60.201 s**	**100,00%**

$$\frac{\#Imagenes\ mostradas}{Tiempo\ total\ de\ ejecucion} = \frac{\#llamadas\ \text{imshow(grayImage);}}{Tiempo\ total\ de\ ejecucion} = \frac{424\ Imagenes}{60.201\ s} = 7.04\ fps$$

Con los resultados obtenidos, es completamente viable la implementacion del algoritmo Viola-Jones en una aplicación cuyo objetivo principal sea el de detectar el parpadeo de un individuo con un rendimiento optimo para aplicarlo en una aplicación de seguridad. Se ha obtenido una buena tasa de imágenes procesadas y una correcta deteccion de los ojos.

Ilustración 21: Representación visual de la detección de ojos con el Algoritmo de Viola-Jones

2.2 Transformada de Hough

La Transformada de Hough circular no es un algoritmo especifico como tal, más bien hay una serie de enfoques diferentes que se pueden tomar en su implementación. Matlab ejecuta dicho algoritmo en tres pasos esenciales.

2.2.1 Búsqueda de círculos de radio desconocido

Este paso se ejecuta con el uso de una matriz acumuladora. Los píxeles de alto gradiente se designan como píxeles candidatos y se les permite emitir 'votos' en la matriz acumulador. En una implementación clásica de CHT, los píxeles candidatos votan en patrón alrededor de ellos que forma un círculo completo de un radio fijo. La Ilustración 22 muestra un ejemplo de píxeles candidatos que se extienden sobre un círculo real (círculo sólido) y el patrón de votación clásico de CHT (círculos de guiones) para los píxeles candidatos.

2.2.2 Estimación del centro

Los votos de los píxeles de los candidatos pertenecientes a un círculo de imágenes tienden a acumularse en el contenedor de la matriz acumuladora correspondiente al centro del círculo. Por lo tanto, los centros del círculo se estiman detectando los picos en la matriz del acumulador. La Ilustración 22 muestra un ejemplo de los píxeles del candidato (puntos sólidos) que yacen en un círculo real (círculo sólido), y sus patrones de votación (círculos de guiones) que coinciden en el centro del círculo real.

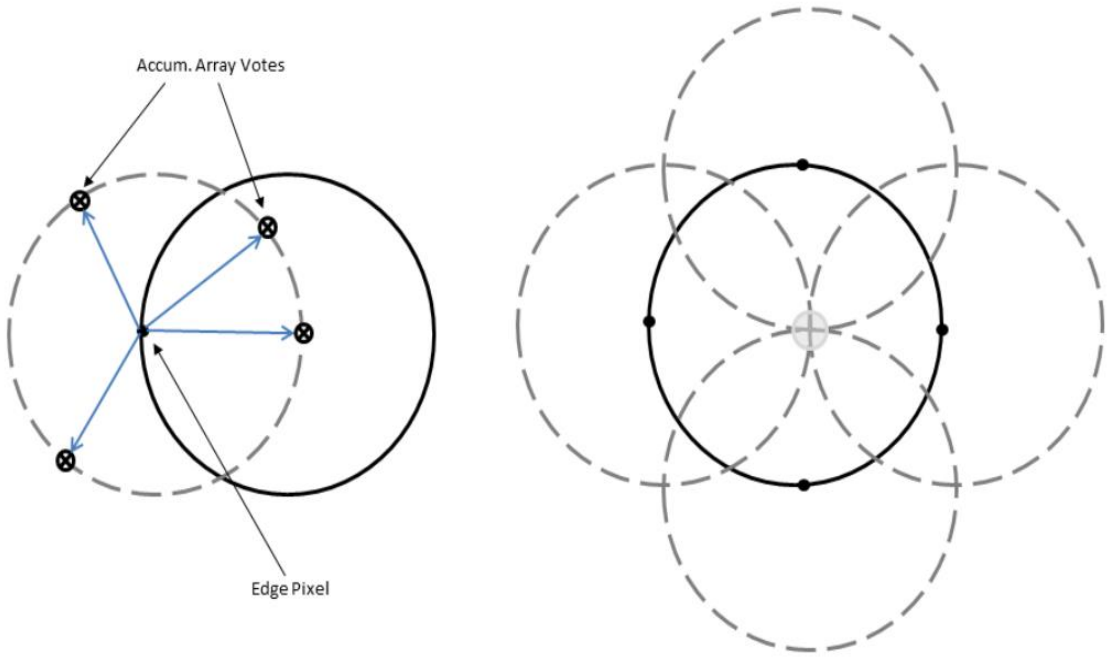


Ilustración 22: Detección de círculos transformada de Hough Matlab

2.2.3 Estimación del radio

Los radios se estiman explícitamente utilizando los centros de círculo estimados junto con la información de la imagen.

2.2.4 Implementación

Tras utilizar la transformada de Hough en Matlab (cuyo código de implementación se puede consultar en el anexo 6.2) para detectar círculos en una imagen procedente de una webcam (cuyas características se pueden consultar en el anexo 6.5) se ha confirmado la alta fiabilidad en la detección de objetos circulares, característica especialmente relevante para el propósito de este proyecto.

La implementación desarrollada en Matlab es bastante sencilla. Se declaran el radio mínimo y máximo del circulo a detectar por él algoritmo, posteriormente se entra en un bucle infinito en donde se procesan las imágenes. Primero se captura con la cámara, la imagen capturada se somete a un conjunto de procesos para adecuarla a la entrada que espera el algoritmo de detección, se procede a hacer la llamada al algoritmo pasándole la imagen previamente procesada, el algoritmo retorna un objeto de detección que contiene las coordenadas exactas en donde se encuentra nuestro objetivo, estos pasos se pueden observar de manera gráfica en el siguiente diagrama de flujo.

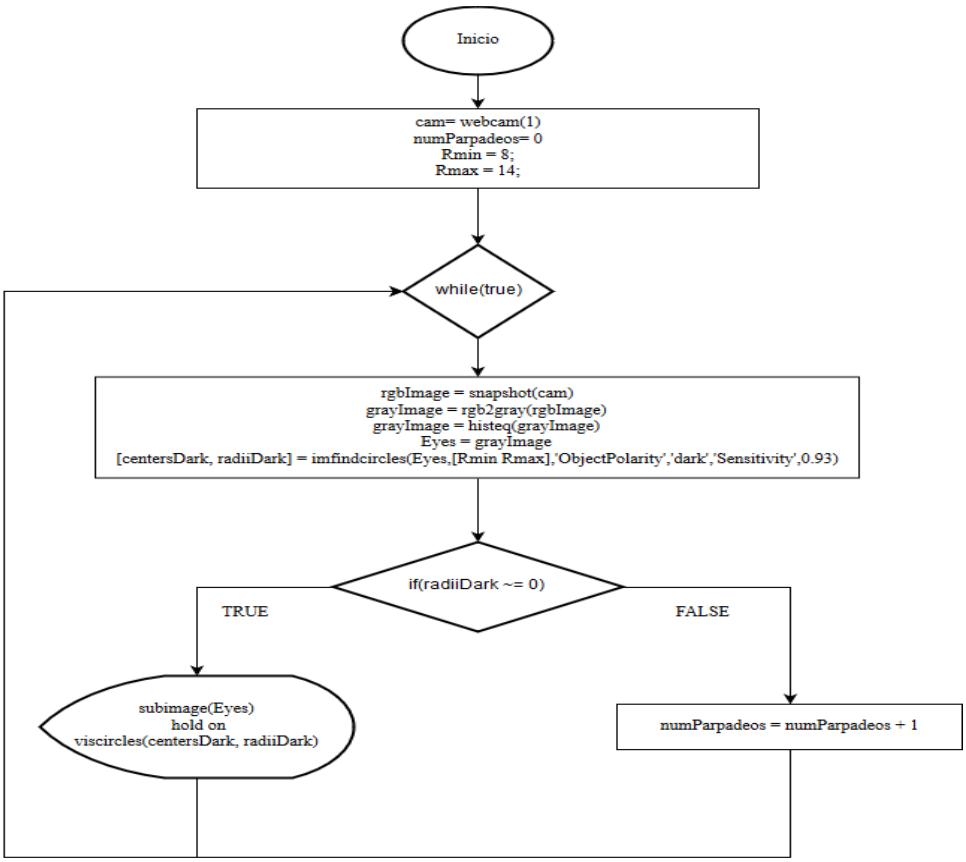


Ilustración 23: Diagrama de flujo algoritmo Transformada de Hough

El algoritmo es capaz de discernir si el ojo se encuentra cerrado o abierto, por contrapartida, este proceso ha detectado una gran cantidad de falsos positivos y aunque técnicamente no todos ellos pueden considerarse como tal, ya que en muchos casos se trata también de elementos circulares, que en la práctica interfieren en la detección inequívoca del círculo del iris. Concluyendo así, que por sí sola, aunque se defina un radio común para la detección del iris, la transformada de Hough no es capaz de discriminar si un círculo detectado, procede de un iris o de cualquier otro objeto circular “parasito” en la imagen.

Ilustración 24: Representación visual de la detección de círculos con la transformada de Hough, pudiéndose ver que aparte de los círculos del iris, detecta otros elementos parásitos

2.3 Viola-Jones + Transformada de Hough

Como se ha observado en la implementación de la transformada de Hough, aplicarla a una imagen sin definir una zona de interés, en nuestro caso los ojos, es totalmente inviable. Para solucionar este problema podemos aprovecharnos de la rápida detección de los ojos que logra el algoritmo de Viola-Jones, y posteriormente aplicar Hough. Explotando los puntos positivos de ambos algoritmos, se puede generar una aplicación con la suficiente fiabilidad en términos de detección y rendimiento para detectar la apertura o cierre de ojos (cuyo código de implementación se puede consultar en el anexo 6.3).

2.3.1 Implementación

Mediante la aplicación del Algoritmo de Viola-Jones, se puede obtener la cara del conductor utilizando el modelo de clasificación 'FrontalFaceCART' , una vez obtenida la cara se pueden discriminar los ojos aplicando otra vez el algoritmo con el modelo de clasificación ‘EyePairSmall’, el siguiente paso es determinar si los ojos se encuentran abiertos o cerrados, para ello se aplica la transformada de Hough a la subregión independiente de cada ojo en busca del círculo del iris, si este es localizado se interpreta que el conductor tiene los ojos abiertos, si no, que los tiene cerrados. Estos pasos se pueden observar de manera gráfica en el siguiente diagrama de flujo.

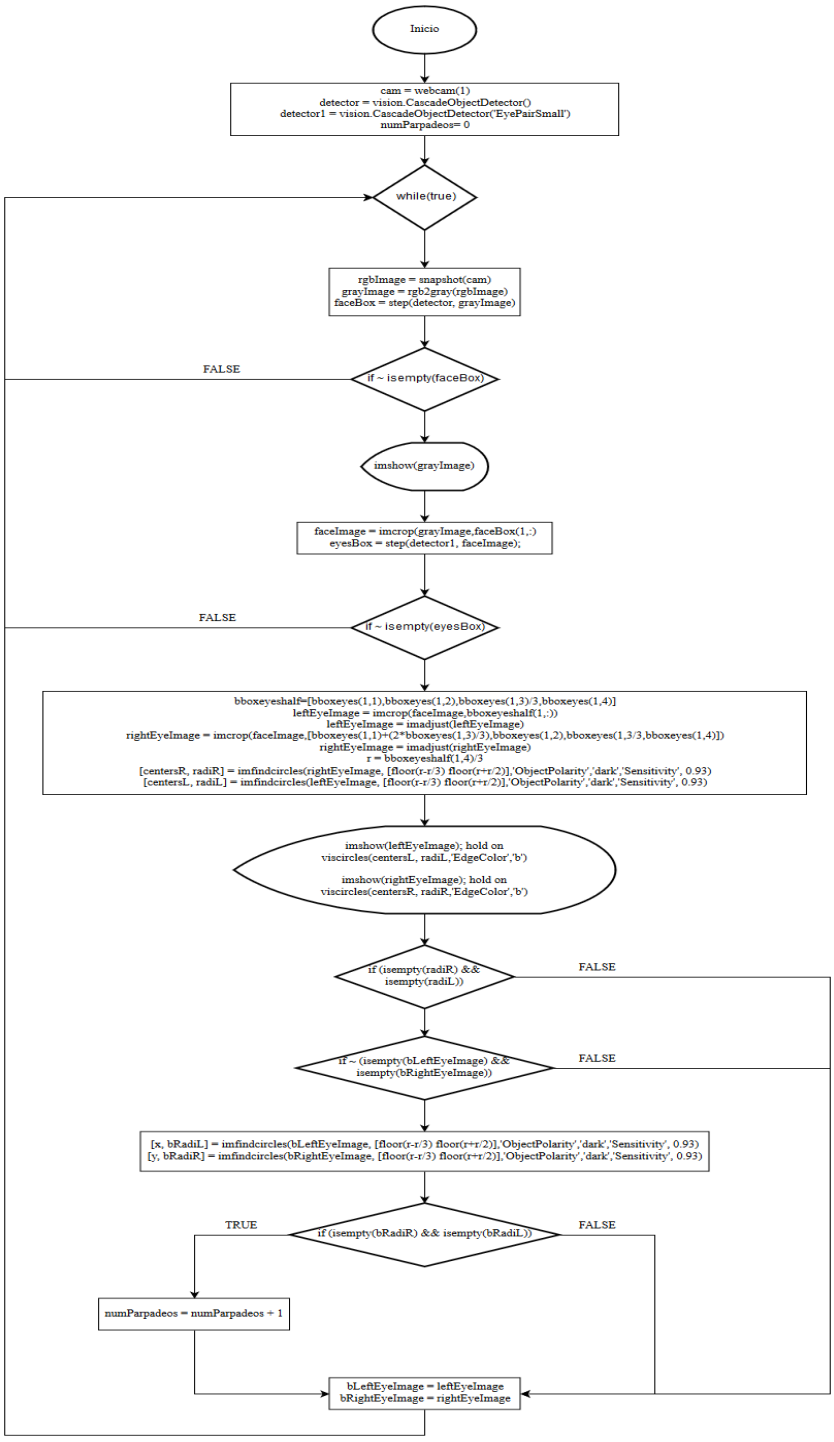


Ilustración 25: Diagrama de flujo algoritmo Viola-Jones + Transformada de Hough

Una de las ventajas de este procedimiento es que el modelo de clasificación 'FrontalFaceCART' permite definir una ventana deslizante más grande que la implementada por defecto, esto acelera considerablemente la detección de la zona de interés. Este procedimiento se aprovecha de la facilidad de detección de diferentes regiones de la cara que aporta el algoritmo de Viola-Jones y de la gran fiabilidad de la capacidad para detectar si el ojo se encuentra abierto o cerrado mediante la detección del iris que aporta la transformada de Hough.

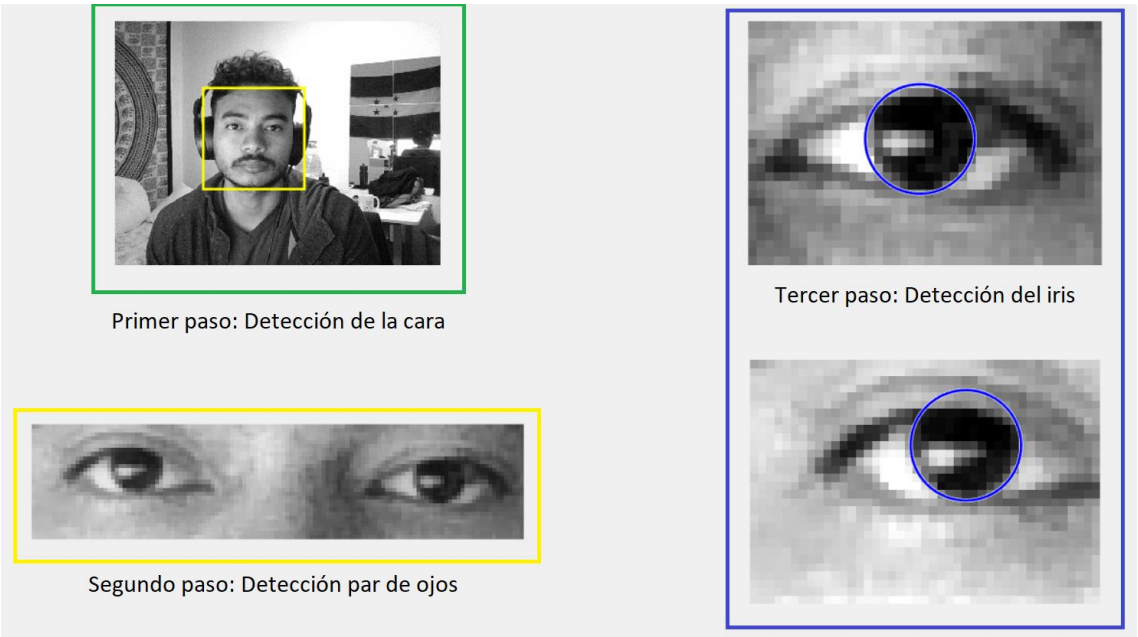


Ilustración 26: Representación visual de la combinación del algoritmo de Viola-jones y la transformada de Hough para la detección de ojos abiertos y ojos cerrados

2.3.2 Coste computacional

El tamaño de la imagen que se ha empleado es de 640x480 pixeles con una ventana de búsqueda para el modelo de clasificación 'FrontalFaceCART' de 150x150 pixeles la cual acelera la detección. Utilizando la herramienta de medida de tiempo de ejecución que proporciona Matlab se obtienen los siguientes resultados:

Número de línea	***Función***	***Llamadas***	***Tiempo total (segundos)***	***% de tiempo***
59	subplot(2,2,2),imshow(rightEye...	299	28.566	33.4%
14	rgbImage = snapshot(cam);	301	12.669	14,8%
19	subplot(2,2,4),imshow(leftEyeI...	299	12.637	14.8%
18	subplot(2,2,1),imshow(grayImag...	301	12.012	14%
15	[centersR, radiR] = imfindcirc...	299	3.185	3.7%
Resto de líneas			16.555	19.3%
Totales			**85.624**	**100,00 %**

$$\frac{\#Imagenes\ mostradas}{Tiempo\ total\ de\ ejecucion} = \frac{\#llamadas\ \text{imshow}();}{Tiempo\ total\ de\ ejecucion} = \frac{301\ Imagenes}{85.614\ s} = 3.5\ fps$$

De esta forma se puede observar que las sentencias que conllevan más tiempo, son aquellas las cuales deben representar imágenes a excepción de la línea "[centersR, radiR] = imfindcirc..." la cual ejecuta la transformada de Hough. Este resultado es muy interesante ya que en realidad la representación de imágenes es simplemente para tener una perspectiva de la visión de la máquina y una forma representativa de lo que se está ejecutando, en caso de que se quiera

aplicar el detector en una aplicación en el mundo real todas estas sentencias se podrían omitir.

Para evaluar el algoritmo sin representar imágenes “complejas” se ha desarrollado una segunda versión en la cual se omite todas aquellas imágenes en tiempo real y solamente se imprimen dos cuadros, uno rojo en caso de no detectar los ojos y otro verde que indica que se está detectando.

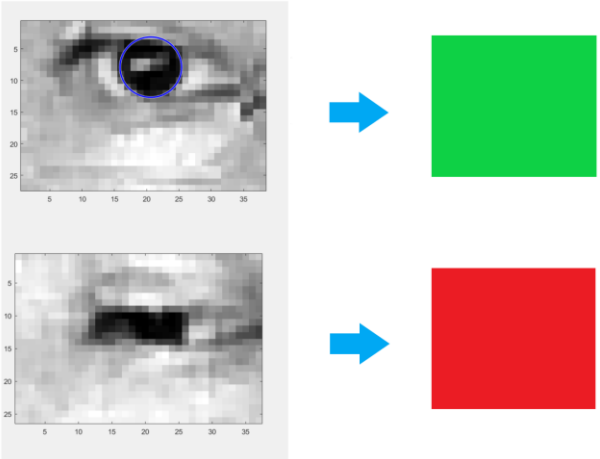


Ilustración 27: Representación booleana de "ojo abierto" en verde y "ojo cerrado" en rojo, con el propósito de ahorrar recursos al no mostrar la imagen a tiempo real

De esta forma se ahorran los recursos utilizados por la máquina para representar imágenes en tiempo real.

Ejecutando la herramienta de medida de tiempo obtenemos los siguientes resultados para la versión simplificada:

Número de línea	***Función***	***Llamadas***	***Tiempo total (segundos)***	***% de tiempo***
45	imshow('EyesDetection_Green.jp...	332	30.591	56.50%
48	imshow('EyesDetection_Red.jpg'...	59	5.191	9.60%
42	[centersL, radiL] = imfindcirc...	391	3.856	7,10%
20	faceBoxeyes = step(detector1, ...	400	3.801	7.0%
41	[centersR, radiR] = imfindcirc...	391	3.755	6.90%
Resto de líneas			6.926	12.8%
Totales			**54.119**	**100,00 %**

$$\frac{\#Imagenes\ procesadas}{Tiempo\ total\ de\ ejecucion} = \frac{\#llamadas\ \text{algoritmo}}{Tiempo\ total\ de\ ejecucion} = \frac{400\ Imagenes}{63.969\ s} = 7.4\ fps$$

El resultado es bastante interesante y claro, como se puede observar las sentencias que mayor tiempo requieren siguen siendo aquellas que deben representar alguna imagen, aunque en este caso sea una imagen que tenemos guardada en local. El siguiente punto a destacar es que ejecutando la versión simplificada el algoritmo ha mejorado considerablemente en el número de fps.

Puntos a destacar:

- Representar imágenes en tiempo real tiene un coste computacional elevado.
- Si se evalúan los algoritmos con imágenes “grandes” sin definir una zona de interés, esto tiene un peso fundamental en lo que tarde la maquina en encontrar los objetos de interés en nuestro caso los ojos.
- Si se simplifica la aplicación sin representación de imágenes en tiempo real se consigue que se mejoren las prestaciones considerablemente (ilustración 25).

2.4 Pruebas con datasets

El objetivo principal de la aplicación, es proporcionar una herramienta cuyas prestaciones sean lo suficientemente seguras para poder convertirse en una opción dentro del mercado de alta fiabilidad, para garantizar dichas prestaciones se ha de testear el algoritmo fuera de los entornos de pruebas que se han utilizado en el desarrollo. Por ello, utilizando un dataset público de 450 imágenes de caras de diferentes individuos se procedió a ver el comportamiento de las implementaciones de los algoritmos viola-jones + transformada de Hough (cuyo código de implementación se puede consultar en el anexo 6.3) y Viola-Jones (cuyo código de implementación se puede consultar en el anexo 6.1).

El dataset utilizado fue recogido por Markus Weber en el Instituto de Tecnología de California. Consta de 450 imágenes de caras, de dimensiones 896 x 592 píxeles y formato JPEG, el cual tiene 27 personas únicas con diferente Iluminación, expresiones y fondos.

2.4.1 Resultados

2.4.1.1 Viola-Jones + Transformada de Hough

De las 450 imágenes, en 37 no se detectaron los ojos y en 16 no se detectó la cara. Como se ha descrito anteriormente, en la implementación del algoritmo se define una ventana de búsqueda más grande que la utilizada por defecto para acelerar la detección de la cara. El inconveniente de este paso es que si el conductor o persona a detectar esta muy alejado de la cámara, el algoritmo no será capaz de encontrar el objeto buscado. Para el objetivo de esta aplicación en cual el conductor debe estar en una posición fija y siempre relativamente cerca de la cámara, no presenta inconvenientes, por este motivo el algoritmo no detecto nada en las 16 imágenes antes mencionadas. Este problema tendría fácil solución, se podría definir una ventana de búsqueda más pequeña, pero conllevaría perder en rendimiento y que la transformada de Hough no fuera capaz de detectar un radio tan pequeño.

Ilustración 28: Imágenes en las cuales no se detectó la cara del individuo

Las 16 imágenes descartadas por el algoritmo no se tendrán en cuenta en el análisis de resultados.

De las 37 imágenes en las cuales no se detectaron los ojos se dividen de la siguiente manera: 5 imágenes con mala iluminación, 8 ojos cerrados o medio cerrados, 23 en las cuales el algoritmo ha fallado y 1 que no corresponde a una persona.

Ilustración 29: Imágenes en las cuales no se detectó el iris

Ilustración 30: Fallo en la detección del iris.

TP	397	Imágenes en las que el algoritmo ha actuado correctamente.
TN	14	Imágenes con ojos cerrados o condiciones de iluminación desfavorables.
FP	0	
FN	23	Imágenes en las que el algoritmo ha fallado
Total	434	

Con los resultados obtenidos para este conjunto de imágenes en específico, el algoritmo fallo en el 5.52% de los casos. Aunque a priori parezca una tasa de error pequeña, observado las imágenes para las que el algoritmo falla en algunas ocasiones no podemos decir que presente una alta fiabilidad.

2.4.1.2 Viola-Jones

De las 450 imágenes en 8 no se detectaron los ojos, las cuales se dividen la siguiente forma: 3 presentas malas condiciones de iluminación, 3 no corresponden a personas, 1 la persona tiene los ojos cerrados y 1 el algoritmo ha fallado.

Ilustración 31: Imágenes en las cuales no se detectan ojos

El algoritmo no fue capaz de detectar el parpadeo en tres imágenes.

Ilustración 32: Imágenes en las cuales se detectan ojos

TP	***439***	Imágenes en las que el algoritmo ha actuado correctamente.
TN	7	Imágenes con ojos cerrados o condiciones de iluminación desfavorables.
FP	3	Imágenes que se ha detectado el ojo cuando este estaba cerrado.
FN	1	Imagen con ojos abiertos que no se detectó.
Total	450	

Con los resultados obtenidos para este conjunto de imágenes en específico, el algoritmo fallo en el 0.88% de los casos. La tasa de error es muy baja pero el algoritmo fallo en identificar el parpadeo.

Aunque los dos algoritmos se desempeñen bien en el entorno de pruebas, al cambiar las imágenes de entrada tienen problemas para hacer la detección correctamente. En el capítulo 3 se procederá a discutir estos resultados.

CAPÍTULO 3. DISCUSÍON DE LOS RESULTADOS

En el capítulo 2, se expone el conjunto teórico que forman los algoritmos de detección de objetos, se parte de la premisa según sus propios creadores, que el algoritmo de Viola-Jones tiene un coste computacional muy bajo y una alta probabilidad de verdaderos positivos, también se dice que la transformada de Hough es una buena alternativa para detectar objetos circulares y, así pues, aprovechar esta característica en la detección del iris.

Este capitulo busca de forma objetiva hacer una comparación en términos de fiabilidad y coste computacional de las dos versiones finales implementadas. Además, se discute la posibilidad de extrapolar los resultados obtenidos a un terminal móvil, mediante una recapitulación de las herramientas de desarrollo disponibles.

3.1 Viola-Jones vs Viola-Jones + Transformada de Hough

Viola Jones	Tamaño imagen de entrada	640x360 pixeles
	Modelos de clasificación	LeftEyeCART - RightEyeCART
	Frame rate	7 fps
Viola Jones + Transformada de Hough	Tamaño imagen de entrada	640x480 pixeles
	Modelos de clasificación	FrontalFaceCART - EyePairSmall
	Frame rate	7.5 fps

Las principales diferencias entre las dos implementaciones son el tamaño de la imagen de entrada, los modelos de clasificación utilizados y la forma que tiene cada implementación en detectar el parpadeo. Una de las ventajas que tiene la implementación conjunta respecto a la implementación desarrollada del algoritmo de Viola-Jones individual, es que podemos definir una ventana de búsqueda mas grande, esto es gracias al modelo de clasificación FrontalFaceCART el cual nos permite configurar este parámetro y de esta forma acelerar considerablemente el proceso de detección.

3.1.1 Comportamiento en situaciones similares

Gran parte de la población utiliza gafas de forma recurrente, por ende, una actividad como la de conducir que casi la totalidad de la información nos llega de manera visual, es normal el uso de lentes por temas de seguridad. Por este motivo el algoritmo implementado debe ser capaz de seguir funcionando en estas condiciones, no se puede privar de la utilización de un elemento casi obligatorio para conductores que lo necesiten.

La siguiente prueba muestra el resultado de las implementaciones con la utilización de gafas.

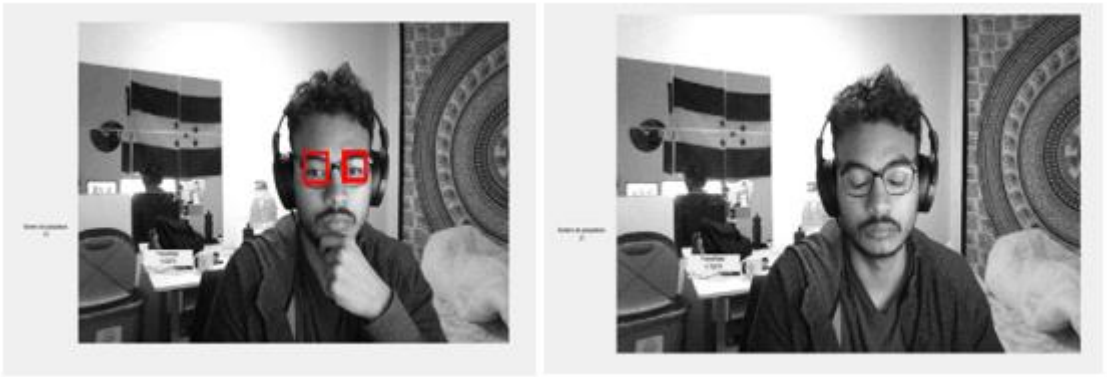


Ilustración 33: funcionamiento del algoritmo Viola-Jones con el uso de gafas

La implementación del algoritmo de Viola-Jones es capaz de seguir su funcionamiento normal con el uso de las gafas, la detección del parpadeo sigue funcionando con normalidad.

Ilustración 34: funcionamiento del algoritmo Viola-Jones + Transformada de Hough con el uso de gafas

No obstante, la implementación del algoritmo conjunto no es capaz de detectar el parpadeo con las gafas. La transformada de Hough es muy sensible al ruido, el uso de gafas distorsiona excesivamente los contornos de la imagen, ilustración 35. En la transformada de Hough se define un margen de radios para los cuales el algoritmo debe actuar, si se introduce un elemento externo como las gafas, el contorno del iris del ojo ya no es tan apreciable, por lo que es normal el mal funcionamiento del algoritmo.

Ilustración 35: Resultado de los contornos de la imagen con el uso de gafas

Con los resultados antes vistos queda de manifiesto que un algoritmo con un bloque de aprendizaje automático (Adaboost) como Viola Jones es mas robusto a posibles interferencias y se adapta mejor a posibles cambios que puedan aparecer en la escena. Estos resultados también se pueden observar en el comportamiento que presentan cuando se someten a prueba con un dataset fuera del entorno de desarrollo.

Cabe recordar que, aunque las dos implementaciones se basen en algoritmos con un bloque Adaboost, la implementación conjunta detecta el parpadeo con Hough, por lo que, las ventajas del bloque de aprendizaje no son utilizables para esta funcionalidad.

3.1.2 Rendimiento

En términos de rendimiento la implementación conjunta presenta mejores resultados, esto es gracias a la búsqueda de zonas de interés, esto simplifica enormemente el numero de operaciones a realizar, si se aplican los modelos de clasificación exclusivamente en la zona en donde puede estar el objeto buscado se ahorra un tiempo de maquina importante.

Se debe tener en cuenta que la imagen capturada y representada por la implementación conjunta es de 640x480 pixeles, imagen más grande que la que se alimenta Viola Jones individual que es de 640x360 pixeles.

Como se ha comentado anteriormente, el modelo de clasificación FrontalFaceCART permite definir el tamaño mínimo y máximo del objeto a buscar, si se acota bien este parámetro se descartan un número importante de ventanas deslizantes (subcapítulo 2.1.3) utilizadas por el algoritmo, aquí yace la clave del ahorro en tiempo de ejecución de la implementación conjunta, a diferencia de el otro algoritmo el cual implementa directamente los modelos LeftEyeCART y RightEyeCART con una ventana de 20x20 pixeles. La implementación conjunta parte del hecho de buscar una cara la cual estará siempre en una posición fija respecto a la cámara, por lo que su valor en numero de pixeles siempre oscilará dentro de los mismos valores, esto permite definir una ventana mucho mas grande lo cual se ve reflejado en el rendimiento del algoritmo.

3.1.3 Posibilidad de extender los resultados a un terminal móvil

Uno de los propósitos de este proyecto, es sentar las bases para que en futuros trabajos se pueda implementar el algoritmo en una aplicación móvil, para ello los resultados obtenidos en Matlab deben poder extenderse a un terminal móvil. Matlab proporciona la herramienta Computer Vision System Toolbox™ la cual permite incorporar funciones y proyectos nativos basados en C++ OpenCV, Android permite utilizar dos métodos de programación para incorporar la biblioteca OpenCV:

Nivel básico: solo implementa la API de Java de OpenCV. OpenCV expone casi toda su funcionalidad a Java (incluida la cámara), por lo que ni siquiera es necesario sumergirse en cosas de NDK (herramienta que permite implementar

código con otros lenguajes nativos como C o C++). Simplemente con agregar una referencia a OpenCV, importa sus clases. Todos los cálculos se realizan en el nivel nativo, por lo tanto, la sobrecarga es igual al costo de una o varias llamadas JNI (Java Native Inteface).

Nivel avanzado: interfaz nativa de OpenCV. Generalmente se usa para el desarrollo profesional, y se considera la forma óptima de desarrollo con OpenCv4Android. Gracias a la ayuda de NDK, Android permite llamar a funciones nativas, lo que significa que puede usar la interfaz original de C++ OpenCV. El enfoque que se describe en el nivel básico, funciona en muchos casos, pero si el proceso de procesamiento es largo, las llamadas JNI pueden consumir demasiado tiempo. En tal caso, puede encapsular toda la funcionalidad en una sola clase de C++ y llamarla solo una vez por fotograma. Lo que también es importante es que puede desarrollar (y depurar) toda la lógica de CV en su máquina host. El código C ++ en OpenCV es multiplataforma, lo que significa que puede transferir su código de escritorio a dispositivos móviles.

Gracias a la gran versatilidad que nos proporciona Android para disponer de múltiples herramientas de desarrollo, la implementación de algoritmos basados en OpenCV es totalmente viable, y gracias a la ayuda de NDK, la ejecución de estos algoritmos sería totalmente eficiente y con tiempos de ejecución que nos garantizarían un funcionamiento en términos de rendimientos parecidos a los que obtuvimos en el entorno de pruebas desarrollado en Matlab.

CAPÍTULO 4. CONCLUSIONES Y TRABAJO FUTURO

Al inicio del trabajo se definieron dos objetivos fundamentales en el desarrollo de este estudio, conseguir una aplicación con un alto rendimiento capaz de funcionar en tiempo real y que presente una fiabilidad a la hora de detectar el parpadeo de un conductor. Gracias a la implementación de los algoritmos de detección de objetos se ha dado con dos soluciones de software que cumplen con los objetivos dentro de un escenario controlado. No obstante, los resultados arrojados fuera de dicho escenario con el uso de un dataset de imágenes públicas, no son muy prometedores, queda palpable que se debe tener en cuesta en todo momento las condiciones del entorno.

Las interferencias según el contexto en donde se ejecuta la aplicación no son las mismas, en consecuencia, no es lo mismo implementar un algoritmo para imágenes con una resolución de 896 x 592 píxeles que uno para imágenes de 640x360 pixeles que además presenten condiciones de iluminación diferentes. Se debe ser extremadamente preciso y meticuloso a la hora de tratar la imagen para acoplarla a unas condiciones optimas antes de aplicar cualquier detector de objetos, como se ha podido ver en capítulos anteriores un algoritmo que implemente técnicas de aprendizaje automático (Viola-Jones) se comporta mejor a este tipo de interferencias que uno que carezca de esta propiedad (transformada de Hough).

En cualquier proceso de reconocimiento de imágenes el paso fundamental es diferenciar claramente todos los elementos que componen la imagen. Definir zonas de interés dentro de un escenario, acota el numero de operaciones que ejecutan los algoritmos de detección, por este motivo, la implementación conjunta Viola-jones + transformada de Hough presentaba un muy buen rendimiento para imágenes considerablemente mas grande, que no un algoritmo que directamente se enfocara en la detección sin previamente definir dicha zona como la implementación propuesta de Viola Jones.

La representación de imágenes es un proceso muy costoso incluso mayor que el de detectar algún objeto, hecho que penaliza el rendimiento de la aplicación. Este estudio es puramente académico, asienta las bases para que en futuros trabajos se tenga una idea general de los inconvenientes y ventajas de cada algoritmo, por ende, se necesita de la representación de imágenes para tener una perspectiva de la visión de la máquina y una forma representativa de lo que se está ejecutando, pero en un entorno real estas sentencias se podrían omitir acto que ahorraría recursos de una forma significativa.

Las soluciones propuestas en este proyecto dentro del escenario de pruebas han respondido bien y han cumplido con los objetivos establecidos, porque presentan un buen rendimiento en cuanto al número de imágenes que son capaces de procesar y detectan cuando una persona parpadea.

Como conclusión general hay varios aspectos que hacen de la detección automática de objetos en imágenes un auténtico reto, los cambios de iluminación, que pueden crear sombras o reflejos, y producir pérdidas

importantes de información, la distancia entre el objeto y la cámara, etc. A pesar de ello con las técnicas actuales y los algoritmos desarrollados en los últimos años, si se conoce el entorno y las características que puede tener o lo cambios que pueda sufrir dicho escenario, se hace viable la implementación de una aplicación que sea capaz de interpretar correctamente la información en una imagen.

4.1 Trabajo futuro

A día de hoy, la mayoría de los vehículos de nueva gama integran cámaras infrarrojas o electrodos en el volante para monitoriza el pulso del conductor, pero cualquier otro automóvil anterior a estas implementaciones podría verse beneficiado de un sistema de alerta ante síntomas de cansancio al volante sin ningún tipo de inversión extraordinaria, simplemente con la instalación de un soporte de smartphone para coches y el uso de cámara frontal del dispositivo enfocada hacia el rostro.

Ilustración 36: Representación de la configuración espacial requerida para el uso de este sistema de alerta a la somnolencia con un smartphone y un soporte para coches

Existen otros signos de fatiga que no se reflejan de forma visual en el conductor pero que podrían mejorar el rendimiento del sistema si se recopilaran y procesaran en paralelo con los descritos hasta ahora. Uno de estos síntomas es la frecuencia cardiaca, representada por el pulso del conductor, sería una información altamente útil si se pudiera adquirir paralelamente con la visual, ya que su análisis revelaría muchos detalles sobre el estado de consciencia del piloto. Hoy en día cualquiera de los brazaletes inteligente o smartwatches es capaz de enviar información sobre el ritmo cardiaco al smartphone a través de una conexión bluetooth y dado el gran éxito comercial de estos dispositivos no se puede descartar su integración futura con el sistema descrito en este proyecto.

CAPÍTULO 5. BIBLIOGRAFÍA

5.1. Referencias

[1] DIRECCIÓN GENERAL DE TRÁFICO. (s.f.). OTROS FACTORES DE RIESGO: EL SUEÑO. Recuperado de www.dgt.es/multimedia/educacion_vial/books/14/book.pdf

[2] Fundación CEA. (2015b, 8 julio). ¿Cuáles son los hábitos de los conductores españoles? Recuperado 10 noviembre, 2018, de https://www.fundacioncea.es/images/estudios/estudio-somnolencia-al-volante.pdf

[3] Díaz Miranda, Viloria Rodríguez, Socorro Llanes y Ferrer Borges, Tonathiuh, José Luis, Raisa, Osmany. *Algoritmo de Viola-Jones para detección de rostros en procesadores gráficos.*

[4] Yi-Qing Wang. An Analysis of the Viola-Jones Face Detection Algorithm.

[5] P. Viola and M. Jones, “Robust real-time object detection,” in International Journal of Computer Vision, 2001.

[6] T. Theocharides et.al, “A Parallel Architecture For Hardware Face Detection,” IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, 2006, pp. 452453.

[7] J. P. Harvey, “GPU Acceleration of Object Classification Algorithms Using NVIDIA CUDA,” Master’s Thesis, Rochester Institute of Technology, Rochester, NY, United States, 2009.

[8] Ren Meng et. al, "Acceleration Algorithm for CUDA-based Face Detection,”2013 International Conference on Signal Processing, Communication and Computing, 2013, pp 1-5.

[9] Simone Frintrop, "A Visual Attention System for Object Detection and Goal-directed Search" Rheinische Friedrich-Wilhelms-Universit¨at Bonn Institut f¨ur Informatik and Fraunhofer Institut f¨ur Autonome Intelligente System

CAPÍTULO 6. ANEXOS

6.1. Algoritmo de Viola-Jones en Matlab

En este subcapítulo se puede leer el código de la implementación en Matlab del algoritmo de Viola-Jones para la detección del par de ojos.

```
clear;
cam= webcam();
cam.Resolution = '640x360';
Left =
vision.CascadeObjectDetector('LeftEyeCART','MergeThreshold',15);
Right =
vision.CascadeObjectDetector('RightEyeCART','MergeThreshold',15);
numParpadeos= 0;

while(true)
    tic
    rgbImage = snapshot(cam);
    grayImage = rgb2gray(rgbImage);
    imshow(grayImage);
    hold on;
    RightEye = Right.step(grayImage);
    LeftEye = Left.step(grayImage);
    t = 1/toc;
    if (~isempty(RightEye) && ~isempty(LeftEye))
        for i = 1:size(LeftEye)
            rectangle('Position',
LeftEye(i,:),'LineWidth',5,'LineStyle', '-','EdgeColor','r');
        end
        for i = 1:size(RightEye)
            rectangle('Position',
RightEye(i,:),'LineWidth',5,'LineStyle', '-','EdgeColor','r');
        end
    else
        if(~isempty(frameAnteriorRightEye) &&
~isempty(frameAnteriorLeftEye))
            numParpadeos = numParpadeos + 1;
            TextLabel = {'Número de parpadeos:',numParpadeos};
            txt2 = uicontrol('Style','text',...
                'Position',[100 300 120 30],...
                'String',TextLabel);
        end
    end
    frameAnteriorRightEye = RightEye;
    frameAnteriorLeftEye = LeftEye;
    FrameRate = {'FrameRate:',t};
    txt1 = uicontrol('Style','text',...
        'Position',[100 350 120 30],...
        'String',FrameRate);
end
```

6.2. Transformada de Hough en Matlab

En este subcapítulo se puede leer el código de la implementación en Matlab de la transformada de Hough para la detección del círculo del iris de los ojos.

```
clear all
clf('reset');
cam= webcam(1);
[user] = memory;
Rmin = 8;
Rmax = 14;
numParpadeos= 0;

while(true)
    tic
    rgbImage = snapshot(cam);
    grayImage = rgb2gray(rgbImage);
    grayImage = histeq(grayImage);
    Eyes = grayImage;
    [centersDark, radiiDark] = imfindcircles(Eyes,[Rmin
Rmax],'ObjectPolarity','dark', ...
        'Sensitivity',0.93);

    if(radiiDark ~= 0)
        subplot(1,1,1);
        subimage(Eyes);
        hold on;
        viscircles(centersDark, radiiDark);
        t = 1/toc;
        frameRate = {'FrameRate:',t};
        txt1 = uicontrol('Style','text',...
            'Position',[100 350 120 30],...
            'String',frameRate);

    else
        numParpadeos = numParpadeos + 1;
        TextLabel = {'Número de parpadeos:',numParpadeos};
        txt2 = uicontrol('Style','text',...
            'Position',[100 300 120 30],...
            'String',TextLabel);
    end

    memoria = {'Memoria algoritmo MB:',user.MemUsedMATLAB/(1024^2)};
    txt3 = uicontrol('Style','text',...
        'Position',[100 250 120 30],...
        'String',memoria);
end
```

6.3. Viola-Jones + Hough en Matlab

En este subcapítulo se puede leer el código comentado de la implementación en Matlab del algoritmo de Viola-Jones, para la detección de la región ocular, junto con la posterior aplicación de la transformada de Hough, en cada sub-ventana ocular individualmente, para la detección del círculo del iris.

```
clear all
clf('reset');
cam=webcam(2);


detector = vision.CascadeObjectDetector();
detector1 = vision.CascadeObjectDetector('EyePairSmall');
[user] = memory
numParpadeos= 0;

while (true)
    tic
    rgbImage = snapshot(cam); %Captura de imagen
    grayImage = rgb2gray(rgbImage); %Convertir la imagen a escala de
grises
    grayImage = flip(grayImage, 2);

    %La funcion Step en matlab llama al "system object" y ejecuta el
    %algoritmo (Viola-Jones). Segun el objeto nos devuelve un
argumento de
    %En este caso en concreto la zona de interes (La cara)

    %IMPORTANTE
    faceBox = step(detector, grayImage); %Deteccion de rostro

    if ~ isempty(faceBox) %Si hemos detectado cara
        faceImage = imcrop(grayImage,faceBox(1,:)); %Recorte de la
zona de interes (Cara)%
        subplot(2,2,1),imshow(grayImage); hold on;
        %Dibujamos el cuadrado amarillo de la cara%
        for i=1:size(faceBox,1)
            rectangle('position', faceBox(i, :), 'lineWidth', 2,
'edgeColor', 'y');
        end

        bboxeyes = step(detector1, faceImage); %Deteccion del par de
ojos%

        if ~ isempty(bboxeyes) %Si hemos detectado ojos%

            %Para una represenrtacion mas comoda nos quedamos con la
itad
            %de la cara. bbxeyes [x y width height] (Zona incluyendo
el par de ojos)

bboxeyeshalf=[bboxeyes(1,1),bboxeyes(1,2),bboxeyes(1,3)/3,bboxeyes(1
,4)];

            %Recortamos los dos ojos
            leftEyeImage = imcrop(faceImage,bboxeyeshalf(1,:));
```

```
            leftEyeImage = imadjust(leftEyeImage);

            rightEyeImage =
imcrop(faceImage,[bboxeyes(1,1)+(2*bboxeyes(1,3)/3),bboxeyes(1,2),bb
oxeyes(1,3)/3,bboxeyes(1,4)]);
            rightEyeImage = imadjust(rightEyeImage);

            %Definimos un radio que sea 1/3 de la altura de imagen
            %(Segun las imagenes mostrada del ojo parece una buena
aproximacion)
            r = bboxeyeshalf(1,4)/3;

            [centersR, radiR] = imfindcircles(rightEyeImage,
[floor(r-r/3) floor(r+r/2)], 'ObjectPolarity','dark', 'Sensitivity',
0.93); % Hough Transform
            [centersL, radiL] = imfindcircles(leftEyeImage,
[floor(r-r/3) floor(r+r/2)], 'ObjectPolarity','dark', 'Sensitivity',
0.93); % Hough Transform

            subplot(2,2,4),imshow(leftEyeImage); hold on;
            viscircles(centersL, radiL,'EdgeColor','b');

            subplot(2,2,2),imshow(rightEyeImage); hold on;
            viscircles(centersR, radiR,'EdgeColor','b');

            %Si no se ha encontrado radio en los ojos se concluye
que estan
            %cerrados por lo que comparamos con el frame anterior en
caso
            %que el fotograma anterior tambien esen los ojos
cerrados no se
            %suma parpadeo por lo contrario si
            if (isempty(radiR) && isempty(radiL))

                if ~ (isempty(bLeftEyeImage) &&
isempty(bRightEyeImage))

                    [x, bRadiL] = imfindcircles(bLeftEyeImage,
[floor(r-r/3) floor(r+r/2)], 'ObjectPolarity','dark', 'Sensitivity',
0.93); % Hough Transform
                    [y, bRadiR] = imfindcircles(bRightEyeImage,
[floor(r-r/3) floor(r+r/2)], 'ObjectPolarity','dark', 'Sensitivity',
0.93); % Hough Transform

                    if (isempty(bRadiR) && isempty(bRadiL))
                        %No sumamos parpadeo ya que tanto el frame
actual
                        %como el anterior no se detecta ojos
                    else

                        numParpadeos = numParpadeos + 1;
                        TextLabel = {'Número de
parpadeos:',numParpadeos};
                        txt2 = uicontrol('Style','text',...
                            'Position',[400 200 120 30],...
                            'String',TextLabel);
                    end
                end
            end
```

```
            bLeftEyeImage = leftEyeImage;
            bRightEyeImage = rightEyeImage;

        end
    end

    hold off;
    memoria = {'Memoria algoritmo MB:',user.MemUsedMATLAB/(1024^2)};
    txt3 = uicontrol('Style','text',...
        'Position',[400 150 120 30],...
        'String',memoria);

    t = 1/toc;
    frameRate = {'FrameRate:',t};
    txt1 = uicontrol('Style','text',...
        'Position',[400 250 120 30],...
        'String',frameRate);
end
```

6.4. Dataset

En este subcapítulo se puede leer el código para detectar los ojos en el dataset público de 250 imágenes.

```
clear all
clf('reset');
List = dir('C:/Users/mantu/Documents/MATLAB/PRUEBA/*jpg');
detector = vision.CascadeObjectDetector();
detector1 = vision.CascadeObjectDetector('EyePairSmall');
numParpadeos= 0;

if exist ('C:/Users/mantu/Documents/nFace','dir') == 7
    rmdir ('C:/Users/mantu/Documents/nFace','s');
end

if exist ('C:/Users/mantu/Documents/nEyed','dir') == 7
    rmdir ('C:/Users/mantu/Documents/nEyed','s');
end

mkdir('C:/Users/mantu/Documents/nFace');
mkdir('C:/Users/mantu/Documents/nEyed');

fDir = 'C:/Users/mantu/Documents/nFace/';
eDir = 'C:/Users/mantu/Documents/nEyed/';

for i = 1 : length (List)

    img =
imread(strcat('C:/Users/mantu/Documents/MATLAB/PRUEBA/',List(i).name
));
    grayImage = rgb2gray (img);
    faceBox = step(detector, grayImage);

    if ~ isempty(faceBox)
```

```
        faceImage = imcrop(grayImage,faceBox(1,:));
        subplot(2,2,1),imshow(grayImage); drawnow;

        bboxeyes = step(detector1, faceImage);

        if ~ isempty(bboxeyes)

bboxeyeshalf=[bboxeyes(1,1),bboxeyes(1,2),bboxeyes(1,3)/3,bboxeyes(1
,4)];

            leftEyeImage = imcrop(faceImage,bboxeyeshalf(1,:));
            leftEyeImage = imadjust(leftEyeImage);

            rightEyeImage =
imcrop(faceImage,[bboxeyes(1,1)+(2*bboxeyes(1,3)/3),bboxeyes(1,2),bb
oxeyes(1,3)/3,bboxeyes(1,4)]);
            rightEyeImage = imadjust(rightEyeImage);

            r = bboxeyeshalf(1,4)/3;
            [centersR, radiR] = imfindcircles(rightEyeImage,
[floor(r-r/3) floor(r+r/2)], 'ObjectPolarity','dark', 'Sensitivity',
0.93);

            [centersL, radiL] = imfindcircles(leftEyeImage,
[floor(r-r/3) floor(r+r/2)], 'ObjectPolarity','dark', 'Sensitivity',
0.93);

            subplot(2,2,4),imshow(leftEyeImage); hold on;
            viscircles(centersL, radiL,'EdgeColor','b');

            subplot(2,2,2),imshow(rightEyeImage); hold on;
            viscircles(centersR, radiR,'EdgeColor','b');

            if (isempty(radiR) && isempty(radiL))

                Dir = strcat (eDir,List(i).name);
                imwrite(faceImage,Dir);
                numParpadeos = numParpadeos + 1;
                TextLabel = {'Número de parpadeos:',numParpadeos};
                txt2 = uicontrol('Style','text',...
                    'Position',[400 200 120 30],...
                    'String',TextLabel);

            end
        end

    else

        Dir = strcat (fDir,List(i).name);
        imwrite(faceImage,Dir);
        subplot(2,2,4), imshow('EyesDetection_Red.jpg');
        subplot(2,2,2), imshow('EyesDetection_Red.jpg');

    end

end
```

6.5. Características de la cámara

En este subcapítulo se puede leer las características de la webcam utilizada para la captura de imágenes en tiempo real.

```
webcam with properties:

                  Name: 'Integrated Webcam'
   AvailableResolutions: {'640x480'  '160x120'  '320x180'  '320x240'  '424x240'  '640x360'}
             Resolution: '640x480'
        WhiteBalanceMode: 'auto'
            WhiteBalance: 4600
            ExposureMode: 'auto'
              Saturation: 64
  BacklightCompensation: 1
               Exposure: -6
               Contrast: 0
             Brightness: 0
                  Gamma: 100
              Sharpness: 2
                    Hue: 0
```